A BOOK APART

Les livres de ceux qui font le web

NO. 1

Jeremy Keith

HTML5 POUR LES WEB DESIGNERS

PRÉFACE DE Jeffrey Zeldman

EYROLLES

L'HTML5 est la spécification HTML la plus longue jamais écrite. C'est également la plus puissante et, en un sens, la plus déroutante. Que doivent en retenir les web designers et les développeurs ? Comment exploiter toute la puissance de l'HTML5 dans les navigateurs actuels ?

Avec beaucoup de style et d'esprit, Jeremy Keith va droit à l'essentiel dans ce guide de l'utilisateur brillant et divertissant et répond à toutes ces questions, exemples clairs et concrets à l'appui.

A BOOK APART

Les livres de ceux qui font le web

www.editions-eyrolles.com
Groupe Eyrolles | Diffusion Geodif

Jeremy Keith

HTML5 POUR LES WEB DESIGNERS

EYROLLES

ÉDITIONS EYROLLES
61, bld Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Traduction autorisée de l'ouvrage en langue anglaise
intitulé *HTML5 FOR WEB DESIGNERS* de Jeremy Keith
(ISBN : 978-0-9844425-0-8), publié par A Book Apart LLC

Adapté de l'anglais par Charles Robert

ISBN : 978-2-212-12861-1

TABLE DES MATIÈRES

AVANT-PROPOS

Quand Mandy Brown, Jason Santa Maria et moi-même avons fondé A Book Apart, un sujet nous préoccupait plus que les autres. Un seul auteur pouvait s'y attaquer.

Rien, pas même l'arrivée des « vraies » polices de caractères ni de CSS3, n'avait agité la communauté du web design standardisé comme l'arrivée imminente de l'HTML5. Née des dissensions provoquées par la lenteur et les politiques du W3C, conçue pour un Web d'applications et pas uniquement de documents, cette nouvelle édition de la lingua franca du Web a à la fois excité, énervé et désorienté la communauté des web designers.

Comme il l'a prouvé avec le DOM et JavaScript, Jeremy Keith possède un talent unique qui lui permet d'illuminer l'HTML5 et d'aller droit à l'essentiel pour les besoins des designers-développeurs. C'est ce qu'il fait dans ce livre, avec le strict nombre de mots et d'images nécessaire.

Il existe d'autres livres traitant de l'HTML5, et il en paraîtra bien d'autres encore. Il y aura des livres techniques de 500 pages pour les développeurs d'applications, ceux dont les besoins ont largement aiguillé le développement de l'HTML5. Il y aura des livres secrets encore plus longs pour les créateurs de navigateurs, abordant des défis techniques que vous et moi n'aurons fort heureusement jamais à envisager.

Mais ce livre-ci est pour vous, vous qui créez du contenu sur le Web, qui utilisez ces balises pour leur sens et la sémantique, vous qui concevez des interfaces et des expériences accessibles. Ce sera votre guide de l'utilisateur d'HTML5. Son but, commun à tous les titres à paraître dans le catalogue de A Book Apart, est de démêler un sujet délicat, et de le faire vite pour mieux retourner au travail.

Jeffrey Zeldman

1 LA PETITE HISTOIRE DU BALISAGE

L'HTML est le langage unificateur du World Wide Web. Avec les simples balises qu'il contient, l'Homme a créé un réseau de documents « hyperliés », étonnamment varié, d'Amazon, eBay et Wikipedia aux blogs et sites personnels consacrés aux chats sosies d'Hitler.

L'HTML5 est la dernière itération en date de cette *lingua franca*. Bien qu'elle constitue le changement le plus ambitieux de notre langue commune, ce n'est pas la première fois que le format HTML est mis à jour. Ce langage évolue depuis sa naissance.

Comme le Web lui-même, l'HyperText Markup Language est une invention personnelle de Sir Tim Berners-Lee. En 1991, il écrivit un document intitulé « HTML Tags » dans lequel il proposa moins de deux douzaines d'éléments pouvant être utilisés pour écrire des pages web.

Sir Tim n'est pas l'inventeur des balises, ces mots compris entre les signes inférieur (<) et supérieur (>) ; celles-ci existaient déjà

dans le format SGML (Standard Generalized Markup Language). Plutôt que d'inventer une nouvelle norme, Sir Tim comprit qu'il était préférable de construire sur les fondations existantes, une tendance qui s'observe encore aujourd'hui dans le développement de l'HTML5.

DE L'IETF AU W3C : LA ROUTE VERS L'HTML 4

L'HTML 1 n'a jamais existé. La première spécification officielle fut l'HTML 2.0, publiée par l'IETF, l'Internet Engineering Task Force. De nombreuses fonctions de cette spécification étaient tirées d'applications existantes. Par exemple, le navigateur web Mosaic de 1994, alors leader du marché, permettait déjà aux auteurs d'incorporer des images dans leurs documents à l'aide d'une balise `<img>`. L'élément `img` apparut plus tard dans la spécification HTML 2.0.

L'l'IETF fut supplanté par le W3C, le World Wide Web Consortium, qui publia les versions ultérieures de la norme HTML sur son site web http://www.w3.org. La deuxième moitié des années 1990 vit une vague de révisions de la spécification, jusqu'à la publication de l'HTML 4.01 en 1999.

À ce moment, l'HTML était à son premier tournant capital.

XHTML 1 : L'HTML EN XML

Après l'HTML 4.01, une nouvelle révision du langage, appelée XHTML 1.0, vit le jour. Le X signifiait « eXtrême », et les développeurs web devaient croiser les bras en prononçant la lettre.

Non, pas vraiment. Il s'agissait du X de « eXtensible », et la gestuelle était tout à fait facultative.

Le contenu de la spécification XHTML 1.0 était identique à celui de l'HTML 4.01 et ne comprenait aucun nouvel élément ou attribut. La seule différence résidait dans la syntaxe du langage.

Là où l'HTML offrait une grande liberté aux auteurs dans la rédaction des éléments et des attributs, le XHTML exigeait que l'on suive les règles du XML, un langage de balisage plus strict sur lequel le W3C fondait la plupart de ses technologies.

Des règles plus strictes n'étaient pas forcément mauvaises. Elles incitaient les auteurs à utiliser un style d'écriture unique. Alors que les balises et les attributs pouvaient précédemment être écrits en majuscules, en minuscules voire en un mélange des deux, les balises et les attributs d'un document XHTML 1.0 devaient être écrits en minuscules pour que celui-ci soit valide.

La publication du XHTML 1.0 coïncida avec l'essor des navigateurs compatibles avec CSS. Comme les web designers soutenaient l'émergence de normes du Web incarnée par le Web Standards Project, le XHTML, avec sa syntaxe plus stricte, semblait être un langage de balisage répondant aux « bonnes pratiques ».

Puis, le W3C publia le XHTML 1.1.

Alors que le XHTML 1.0 n'était que du HTML reformulé en XML, le XHTML 1.1 était du XML, du vrai, de l'authentique. Cela signifie qu'il ne pouvait pas être traité avec un type MIME `text/html`. Pourtant, si un auteur publiait un document avec un type MIME XML, le navigateur web le plus populaire du moment, Internet Explorer, ne pouvait pas l'interpréter.

C'était comme si le W3C avait perdu pied avec la réalité quotidienne de la publication sur le Web.

XHTML 2 : SAUVE QUI PEUT !

Si le personnage de Dustin Hoffman dans *Le Lauréat* avait été web designer, le W3C lui aurait dit un mot, juste un mot : XML.

Pour le W3C, l'HTML était fini depuis la version 4. Il commença à développer le XHTML 2, censé conduire le Web vers un nouvel horizon radieux, avec le XML pour décor.

Même si le nom XHTML 2 ressemblait à s'y méprendre à XHTML 1, ces deux langages n'avaient pas grand chose en commun. À la différence du XHTML 1, le XHTML 2 était incompatible avec le contenu existant du Web, et même avec les versions précédentes de l'HTML. Ce devait être un langage pur, affranchi du passé trouble des spécifications précédentes.

Ce fut un désastre.

LE SCHISME : WHATWG TF ?

Une rébellion se dessina au sein du W3C. On aurait dit que le consortium formulait des normes théoriquement pures sans aucun rapport avec les besoins des web designers. Les représentants d'Opera, d'Apple et de Mozilla étaient frustrés par la tournure des événements. Ils voulaient que la priorité soit accordée aux formats permettant la création d'applications web.

Le différend atteignit son paroxysme lors d'un séminaire, en 2004. Ian Hickson, qui travaillait alors pour Opera Software, proposa de développer l'HTML pour permettre la création d'applications web. Sa proposition fut rejetée.

Les rebelles désabusés formèrent leur propre groupe : le Web Hypertext Application Technology Working Group, ou WHATWG pour faire court.

DE WEB APPS 1.0 À L'HTML5

Dès le départ, le WHATWG prit une toute autre direction que le W3C. Le W3C cherche le consensus : des questions sont soulevées, débattues, puis votées. Au WHATWG, des questions sont aussi soulevées et débattues, mais la décision finale, en ce qui concerne le contenu d'une spécification, revient à l'éditeur. Cet éditeur, c'est Ian Hickson.

À première vue, la procédure du W3C semble plus juste et plus démocratique. Dans les faits, les opinions divergentes et les querelles internes peuvent freiner les avancées. Au WHATWG, où tout le monde est libre de contribuer mais où l'éditeur a le dernier mot, les choses vont plus vite. D'ailleurs, l'éditeur ne dispose pas vraiment d'un pouvoir absolu : un comité de pilotage recruté sur invitation peut opposer son veto, dans le cas improbable d'un scénario à la Docteur Folamour.

Au début, le gros du travail du WHATWG portait sur deux spécifications : Web Forms 2.0 et Web Apps 1.0. Ces deux spécifications étaient destinées à compléter l'HTML. Avec le temps, elles furent regroupées dans une seule spécification, simplement appelée HTML5.

LA RÉUNIFICATION

Pendant que le WHATWG développait l'HTML5, le W3C continuait à travailler sur le XHTML 2. Il serait inexact de dire qu'il fonçait droit dans le mur ; il y allait très, très lentement.

En octobre 2006, Sir Tim Berners-Lee admit sur son blog que la tentative de migration du Web de l'HTML vers le XML ne marchait tout bonnement pas. Quelques mois plus tard, le W3C établissait une nouvelle charte créant un groupe de travail HTML. Au lieu de partir de zéro, ils décidèrent judicieusement d'utiliser le travail du WHATWG comme base pour les versions futures de l'HTML.

Toutes ces allées et venues rendirent la situation confuse. Le W3C travaillait simultanément sur deux langages de balisage différents et incompatibles : le XHTML 2 et l'HTML 5 (notez l'espace avant le chiffre cinq). Au même moment, une autre organisation, le WHATWG, travaillait sur une spécification appelée HTML5 (sans espace) qui devait servir de base à l'une des spécifications du W3C !

Un web designer désireux de comprendre la situation aurait mieux fait de s'envoyer l'œuvre complète de David Lynch.

LE XHTML EST MORT : VIVE LA SYNTAXE XHTML !

Ce brouillard de confusion commença à se dissiper en 2009. Le W3C annonça que la charte du XHTML 2 ne serait pas renouvelée. Dans les faits, le format était mort depuis plusieurs années ; cette annonce n'était rien de plus qu'un certificat de décès.

Curieusement, plutôt que de passer inaperçue, la mort du XHTML 2 fut accueillie par la jubilation de quelques mauvais esprits. Les détracteurs du XML se servirent de cette annonce pour tourner en ridicule quiconque avait jamais utilisé le XHTML 1, même si le XHTML 1 et le XHTML 2 n'avaient pratiquement rien en commun

Parallèlement, les auteurs qui utilisaient le XHTML 1 pour appliquer un style d'écriture plus strict craignirent que l'avènement de l'HTML5 ne présageât un retour au désordre.

Comme vous le verrez bientôt, ce n'est pas forcément le cas. L'HTML5 est aussi désordonné ou strict que vous le souhaitez.

L'HISTORIQUE DE L'HTML5

L'état actuel de l'HTML5 n'est plus aussi confus qu'il a été, mais il n'est toujours pas parfaitement clair.

Deux groupes travaillent sur l'HTML5. Le WHATWG crée une spécification HTML5 en suivant une procédure dite de « Commit-Then-Review » : les changements sont appliqués avant d'être examinés et débattus. Le groupe de travail HTML du W3C prend cette même spécification en suivant la procédure inverse (« Review-Then-Commit »). Comme vous pouvez l'imaginer, l'alliance est ardue. Toutefois, il semble que la question épineuse de l'espace ait trouvé un consensus (c'est HTML5 sans espace, si vraiment ça vous intéresse).

La question la plus troublante pour les web designers qui trempent leurs orteils dans les eaux troubles de l'HTML5 est peut-être celle-ci : « Quand sera-t-il prêt ? »

Dans un entretien, Ian Hickson a déclaré que selon lui, l'HTML5 obtiendrait le statut de proposition de recommandation en 2022. Cette annonce a déclenché une vague de protestations de la part de quelques web designers. Ils ne comprenaient pas le sens de « proposition de recommandation », en revanche ils savaient qu'ils n'avaient pas assez de doigts pour compter les années jusqu'en 2022.

Ces protestations étaient indues. Dans ce cas précis, le statut de « proposition de recommandation » requiert deux implémentations complètes de l'HTML5. Au vu de l'envergure de la spécification, cette date est incroyablement ambitieuse. Après tout, les navigateurs n'ont pas les meilleurs antécédents quant à l'implémentation des normes existantes. Il a fallu plus d'une décennie pour qu'Internet Explorer supporte l'élément `abbr`.

La date vraiment importante pour l'HTML5 est 2012. C'est en 2012 que la spécification doit devenir « recommandation candidate », c'est-à-dire être fin prête dans le discours normatif.

Mais cette date n'est pas particulièrement pertinente non plus pour les web designers. Ce qui compte avant tout, c'est la compatibilité des navigateurs avec les nouvelles fonctionnalités. On a commencé à utiliser des morceaux de CSS 2.1 dès que certains navigateurs ont été en mesure de les interpréter. Si l'on avait attendu que tous les navigateurs soient entièrement compatibles avec CSS 2.1 avant de l'utiliser, on serait encore en train d'attendre.

Cela vaut également pour l'HTML5. On ne pourra pas le déclarer « prêt à l'emploi » à un moment précis. On commencera plutôt à utiliser des morceaux de la spécification au gré de leur implémentation dans les navigateurs.

Souvenez-vous, l'HTML5 n'est pas un langage complètement nouveau, parti de rien. C'est une évolution, plus qu'une

révolution, dans l'histoire ininterrompue des langages de balisage. Si vous créez actuellement des sites web avec n'importe quelle version de l'HTML, vous utilisez déjà l'HTML5.

2 LES PRINCIPES DE L'HTML5

La Révolution française fut une ère de bouleversements politiques et sociaux. La ferveur révolutionnaire s'appliqua même à l'heure : pendant une brève période, la République française mit en place un système horaire décimal, chaque jour étant divisé en dix heures et chaque heure en cent minutes. C'était un système parfaitement logique et clairement supérieur au système sexagésimal.

Le temps décimal fut un échec. Personne ne l'utilisa. On pourrait dire la même chose du XHTML 2. Le W3C a redécouvert l'histoire de la France postrévolutionnaire : les comportements sont peu enclins au changement.

PRINCIPES DE CONCEPTION

Désireux d'éviter les erreurs du passé, le WHATWG a rédigé une liste de principes de conception afin d'orienter le développement de l'HTML5. Un des principes clés consiste à « supporter le contenu existant ». Cela signifie qu'il n'y a pas d'an zéro pour l'HTML5.

Là où le XHTML 2 avait tenté de balayer tout ce qui précédait, l'HTML5 se construit à partir de spécifications et d'implémentations existantes. L'essentiel de l'HTML 4.01 a survécu dans l'HTML5.

L'HTML5 comprend d'autres principes, tels que « ne pas réinventer la roue » et « paver le sentier des vaches ». Cela veut dire que s'il existe une façon répandue d'accomplir une tâche chez les web designers, et même si ce n'est pas la meilleure, elle doit être codifiée en HTML5. On pourrait aussi dire : « si ce n'est pas cassé, on ne répare pas ».

Nombre de ces principes de conception vous seront familiers si vous vous êtes déjà intéressé à la communauté des microformats (http://microformats.org). La communauté HTML5 partage la même approche pragmatique, consistant à développer un format sans trop se soucier des problèmes théoriques.

Cette attitude est consacrée par le principe de « priorité des circonscriptions », qui dicte : « En cas de conflit, privilégiez d'abord les utilisateurs, puis les auteurs, puis les implémenteurs, puis les spécificateurs, et enfin la pureté théorique. »

Ian Hickson a déclaré à de nombreuses reprises que ceux qui décidaient vraiment du contenu de l'HTML5 étaient les créateurs de navigateurs. En effet, si le créateur d'un navigateur refuse de supporter une proposition particulière, il est inutile d'ajouter cette proposition à la spécification puisque celle-ci ne serait que fiction. D'après le principe de priorité des circonscriptions, nous autres web designers avons une voix plus forte encore : si nous refusons d'utiliser une partie de la spécification, celle-ci est tout aussi fictive.

SOYONS PRAGMATIQUES

La création de l'HTML5 est mue par une tension interne constante. D'un côté, la spécification doit être assez puissante pour supporter la création d'applications web ; de l'autre, l'HTML5 doit être

compatible avec le contenu existant, même si celui-ci est, en grande partie, un foutoir sans nom. Si la spécification s'égare trop loin dans une direction, elle subira le même sort que le XHTML 2, mais si elle va trop loin dans l'autre, elle ne fera qu'entériner les balises `<font>` et les tableaux, qui constituent l'essentiel de la mise en page de nombreuses pages web.

Il s'agit d'un équilibre fragile qui requiert une approche pragmatique et pondérée.

GESTION DES ERREURS

La spécification HTML5 ne se borne pas à déclarer comment les navigateurs doivent traiter les balises bien formées. Pour la première fois, une spécification définit également comment les navigateurs doivent se comporter avec les documents mal formés.

Jusqu'à maintenant, les créateurs de navigateurs devaient déterminer individuellement la façon de gérer les erreurs. Cela consistait généralement à faire de l'ingénierie inverse à partir du comportement du navigateur le plus populaire, soit une vraie perte de temps. Les créateurs de navigateurs feraient mieux d'implémenter de nouvelles fonctionnalités au lieu de perdre leur temps à dupliquer les solutions des concurrents.

Définir la gestion des erreurs dans HTML5 est une tâche incroyablement audacieuse. Même si l'HTML5 disposait des mêmes éléments et attributs que l'HTML 4.01, sans nouvelle fonctionnalité, définir la gestion des erreurs d'ici à 2012 serait une tâche sisyphéenne.

La gestion des erreurs ne présente peut-être pas beaucoup d'intérêt pour les web designers, surtout si l'on écrit des documents valides et bien formés dès le départ, mais elle est très importante pour les créateurs de navigateurs. Alors que les spécifications des langages précédents étaient écrites pour les auteurs, l'HTML5 est écrit pour les auteurs *et* les implémenteurs. Gardez cela à l'esprit quand vous parcourrez la spécification. C'est la raison pour laquelle la spécification HTML5 est si épaisse

et semble avoir été écrite avec un niveau de détail normalement réservé aux philatélistes ou aux champions d'échecs.

C'EST GRAVE, DOCTYPE ?

Une déclaration de type de document, ou doctype, est traditionnellement utilisée pour spécifier le type de balisage du document.

Le doctype de l'HTML 4.01 ressemble à ceci (le symbole » indique que la ligne se poursuit) :

```
<!DOCTYPE HTML PUBLIC »
"-//W3C//DTD HTML 4.01//EN" »
"http://www.w3.org/TR/html4/strict.dtd">
```

Voici le doctype du XHTML 1.0 :

```
<!DOCTYPE html PUBLIC »
"-//W3C//DTD XHTML 1.0 Strict //EN" »
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Ces déclarations sont difficilement compréhensibles pour vous et moi, mais elles ne font que dire à leur façon « ce document est écrit en HTML 4.01 », ou « ce document est écrit en XHTML 1.0 ».

On pourrait penser que le doctype qui déclare « ce document est écrit en HTML5 » comporte le numéro cinq quelque part, mais ce n'est pas le cas. Le doctype de l'HTML5 ressemble à cela :

```
<!DOCTYPE html>
```

Il est si simple que même moi, je pourrais m'en souvenir.

Mais c'est de la folie ! Sans numéro de version dans le doctype, comment allons-nous spécifier les versions futures de l'HTML ?

La première fois que j'ai vu le doctype de l'HTML5, j'ai pensé que c'était le comble de l'arrogance. Je me suis dit : « Pensent-ils vraiment qu'il s'agit là de la dernière spécification d'un langage de balisage jamais écrite ? »

Une fois de plus, des gens semblaient convaincus qu'ils inauguraient une nouvelle ère.

En réalité, le doctype de l'HTML5 est très pragmatique. Puisque l'HTML5 doit supporter le contenu existant, le doctype doit pouvoir s'appliquer à un document en HTML 4.01 ou en XHTML 1.0. Toutes les versions futures de l'HTML devront également supporter le contenu existant de l'HTML5 ; l'idée même d'ajouter un numéro de version à un document est donc viciée.

La vérité, c'est que les doctypes ne sont pas vraiment importants. Admettons que vous créiez un document avec un doctype HTML 4.01. Si ce document comprend un élément d'une autre spécification, comme l'HTML 3.2 ou l'HTML5, les navigateurs interpréteront quand même cette partie du document. Les navigateurs supportent les fonctionnalités, et non les doctypes.

Les déclarations de type de document étaient, à la base, destinées aux validateurs et non aux navigateurs. Un navigateur ne prêtera attention au doctype qu'en cas de changement de doctype, un petit bidouillage ingénieux permettant d'alterner entre les modes d'affichage « quirks » et « standards », suivant la présence d'un doctype adéquat.

Le doctype HTML5 constitue le minimum requis pour garantir que les navigateurs affichent une page en mode *standard*. En fait, c'est la seule et unique raison d'inclure ce doctype. Un document HTML écrit sans le doctype HTML5 peut quand même être un document HTML5 valide.

LA SIMPLICITÉ MÊME

Le doctype n'est pas la seule chose à avoir été simplifiée dans l'HTML5.

Si vous souhaitez spécifier l'encodage des caractères d'un document balisé, la meilleure manière est de vous assurer que votre serveur envoie l'en-tête `Content-Type` adéquat. Si vous voulez en être doublement certain, vous pouvez également spécifier le jeu de caractères à l'aide d'une balise `<meta>`. Voici la déclaration `meta` pour un document écrit en HTML 4.01 :

```
<meta http-equiv="Content-Type" content="text/html; »
charset=UTF-8">
```

Voici une manière bien plus simple de faire la même chose en HTML5 :

```
<meta charset="UTF-8">
```

Comme pour le doctype, cette déclaration d'encodage simplifiée compte le nombre minimum de caractères requis pour être correctement interprétée par les navigateurs.

La balise `<script>` pouvait également se permettre de perdre un peu de poids. On y ajoute couramment un attribut `type` avec la valeur « text/javascript ».

```
<script type="text/javascript" src="file.js"></script>
```

Les navigateurs n'ont pas besoin de cet attribut. Ils partiront du principe que le script est écrit en JavaScript, le langage de script le plus populaire du Web (et soyons honnêtes : le *seul* langage de script du Web) :

```
<script src="file.js"></script>
```

De même, il est inutile de donner la valeur « text/css » à l'attribut `type` pour appeler un fichier CSS :

```
<link rel="stylesheet" type="text/css" href="file.css">
```

Vous pouvez simplement écrire :

```
<link rel="stylesheet" href="file.css">
```

SYNTAXE : BALISER LE SENTIER

Certains langages de programmation, comme Python, imposent une façon particulière d'écrire des instructions : il est obligatoire d'indenter le code. D'autres langages de programmation, comme le JavaScript, ne prêtent pas attention au formatage ; l'indentation est donc facultative.

Si vous cherchez à passer une soirée divertissante à moindre frais, mettez une bande de programmeurs dans la même pièce et prononcez les mots « indentation obligatoire » Délectez-vous des heures de *flaming* qui s'ensuivent.

Une question philosophique fondamentale est au cœur du débat sur l'indentation : un langage doit-il imposer un style d'écriture particulier, ou les auteurs doivent-ils être libres d'écrire dans le style qui leur convient ?

Dans les langages de balisage, l'indentation est optionnelle. Si vous souhaitez aller à la ligne et ajouter une indentation à chaque fois que vous insérez un élément, grand bien vous fasse, mais les navigateurs et les validateurs n'en ont pas besoin. Cela ne veut pas dire que le balisage est une grande foire d'empoigne. Certains langages de balisage imposent un style d'écriture plus strict que d'autres.

Avant le XHTML 1.0, la casse était sans importance pour le format des balises. Il n'était pas obligatoire de placer les attributs entre guillemets. On pouvait même ne pas fermer certaines balises.

Le XHTML 1.0 applique la syntaxe du XML. Toutes les balises doivent être écrites en minuscules, tous les attributs doivent être délimités par des guillemets, et tous les éléments doivent comprendre une balise de fermeture. Dans le cas particulier des éléments autonomes comme `br`, la balise de fermeture obligatoire est remplacée par une barre de fermeture oblique : `<br />`.

Avec l'HTML5, tout est possible. Majuscules, minuscules, avec ou sans guillemets, balise auto-fermante ou non : c'est à vous de décider.

J'utilise le doctype XHTML 1.0 depuis des années. J'aime le fait de devoir écrire dans un style particulier, et j'aime la façon dont le validateur W3C applique ce style. Maintenant que j'utilise l'HTML5, c'est à moi de choisir le style qui me convient.

Je comprends pourquoi certaines personnes redoutent les approximations de la syntaxe de l'HTML5. Elles ont l'impression de tourner le dos à des années de bonnes pratiques. Certains disent même que la syntaxe laxiste de l'HTML5 favorise le mauvais balisage. Je ne pense pas que ce soit vrai, mais je comprends cette inquiétude. C'est un peu comme si un langage de programmation qui appliquait l'indentation obligatoire devenait subitement plus indulgent.

Pour ma part, je n'ai pas de problème avec la syntaxe décontractée de l'HTML5. J'ai fini par assumer le fait de devoir appliquer moi-même mon propre style d'écriture préféré, mais j'aimerais voir plus d'outils me permettant de l'évaluer par rapport à un style particulier. Dans le monde de la programmation, cela s'appelle un outil lint : un programme signalant les pratiques de programmation suspectes. Un outil lint pour le balisage serait différent d'un validateur, qui compare le code à un doctype. L'idéal serait de pouvoir combiner les deux outils en une super machine à tout faire.

Quiconque programmera un tel appareil gagnera le respect et l'admiration éternels de tous les web designers.

ON N'UTILISE PAS CE GENRE DE LANGAGE

Dans les versions précédentes de l'HTML, le procédé consistant à supprimer de la spécification un élément ou un attribut existant auparavant était appelé dépréciation. On conseillait aux web designers de ne pas utiliser d'éléments dépréciés, de ne pas leur envoyer de carte de vœux à Noël, ni même de les évoquer pendant les repas de famille.

Il n'y a pas d'éléments ni d'attributs dépréciés dans l'HTML5, mais il y a de nombreux éléments et attributs *obsolètes*.

Non, ce n'est pas un abus de politiquement correct. Il existe une subtile différence de sens entre « obsolète » et « déprécié ».

Puisque l'HTML5 vise à être compatible avec le contenu existant, la spécification doit reconnaître les éléments existant auparavant, même quand ceux-ci ne font plus partie de l'HTML5. On aboutit à une situation légèrement confuse, où la spécification dit simultanément « auteurs, n'utilisez pas cet élément » et « navigateurs, voici comment interpréter cet élément ». Si l'élément était déprécié, il ne serait pas mentionné du tout dans la spécification, mais puisqu'il est obsolète, il est inclus pour les besoins des navigateurs.

À moins que vous ne fabriquiez un navigateur, vous pouvez traiter les éléments et les attributs obsolètes de la même façon que les éléments et les attributs dépréciés : ne les utilisez pas dans vos pages web et ne les invitez pas pour prendre l'apéro.

Si vous tenez à utiliser un élément ou un attribut obsolète, votre document sera « non conforme ». Les navigateurs l'interpréteront convenablement, mais vous serez la risée du Web.

Adieu, content de t'avoir connu

Les éléments `frame`, `frameset` et `noframes` sont rendus obsolètes. Ils ne nous manqueront pas.

L'élément `acronym` est rendu obsolète, nous faisant ainsi gagner des années de discussion qui seraient plus utilement dépensées à calculer le taux de disparition des petites cuillères dans les restaurants d'entreprise. Ne pleurez pas l'élément `acronym` ; utilisez simplement l'élément `abbr`. Oui, je sais qu'il existe une différence entre les acronymes et les abréviations : les acronymes se prononcent comme un seul mot, à l'instar de l'OTAN ou du laser. Rappelez-vous seulement que tous les acronymes sont des abréviations, mais que toutes les abréviations ne sont pas des acronymes.

Les éléments de présentation, comme `font`, `big`, `center` et `strike` sont rendus obsolètes en HTML5. En réalité, ils sont obsolètes depuis des années ; il est bien plus simple d'obtenir les mêmes effets de présentation à l'aide de propriétés CSS comme `font-size` et `text-align`. De même, les attributs de présentation comme `bgcolor`, `cellspacing`, `cellpadding` et `valign` sont obsolètes. Utilisez plutôt CSS.

Tous les éléments de présentation ne sont pas obsolètes. Certains ont suivi un programme de rééducation et on leur a donné une nouvelle chance.

TURN AND FACE THE STRANGE (CH-CH-CHANGES)

L'élément `big` est obsolète, mais pas l'élément `small`. Cette incohérence apparente a été résolue en redéfinissant `small`. Il a perdu sa fonction stylistique « afficher ce texte en petite taille ». Il possède maintenant la valeur sémantique « voici des petits caractères », pour le jargon juridique ou les conditions d'utilisation.

Évidemment, neuf fois sur dix, vous voudrez afficher les petits caractères en petite taille. Il faut simplement comprendre que l'aspect purement cosmétique de l'élément a été supplanté.

Avant, l'élément `b` voulait dire « afficher ce texte en gras ». Il est maintenant utilisé pour qu'une portion de texte soit « stylistiquement décalée de la prose normale sans exprimer une importance supplémentaire ». Si le texte est effectivement plus important, l'élément `strong` sera plus approprié.

De même, l'élément `i` ne veut plus dire « afficher ce texte en italique ». Il signifie que le texte est « dans une voix ou une humeur alternative ». Une fois de plus, l'élément n'implique pas d'importance particulière ou d'emphase. Pour l'emphase, utilisez l'élément `em`.

Ces changements peuvent sembler jouer sur les mots. C'est effectivement le cas, mais ils aident également à améliorer

l'universalité de l'HTML5. Si vous réfléchissez aux mots « gras » et « italique », ceux-ci n'ont de sens que pour un support visuel comme un écran ou une page. En enlevant l'aspect visuel de la définition des éléments, la spécification reste utile pour les *user agents* non visuels, tels que les lecteurs d'écran. Cela encourage également les designers à penser au-delà des environnements de rendu visuel.

Illi-cite

L'élément `cite` a été redéfini dans l'HTML5. Alors qu'il désignait précédemment une « référence à d'autres sources », il signifie maintenant « le titre d'une œuvre ». Bien souvent, la référence citée sera le titre d'un livre ou d'un film, mais la source peut tout aussi bien être une personne. Avant l'HTML5, on pouvait baliser le nom de cette personne avec `cite`. C'est maintenant expressément interdit, et tant pis pour la rétrocompatibilité.

La justification de cette œuvre de révisionnisme tient dans ce raisonnement : les navigateurs mettent en italique le texte situé entre deux balises `<cite>` ; le titre d'une œuvre est généralement en italique ; le nom d'une personne n'est généralement pas en italique ; donc l'élément `cite` ne doit pas être utilisé pour baliser le nom d'une personne.

C'est trop injuste. Je suis favorable à ce que l'HTML5 s'inspire des navigateurs, mais là, c'est la queue qui remue le chien.

Heureusement, aucun validateur ne peut déterminer si le texte situé entre les balises `<cite>` fait référence à une personne ou non. Rien ne nous empêche donc d'utiliser l'élément `cite` de manière sensée et rétrocompatible.

L'élément a gonflé aux hormones

Alors que certains changements jouent franchement sur les mots, un élément subit une transformation radicale avec l'HTML5.

L'élément `a` est, sans conteste, l'élément le plus important de l'HTML. Il transforme notre texte en hypertexte. Il forme le tissu conjonctif du World Wide Web.

L'élément `a` a toujours été un élément de type en-ligne (*inline*). Si vous vouliez créer un lien à partir d'un titre et d'un paragraphe, vous deviez utiliser plusieurs éléments `a` :

```
<h2><a href="/about">About me</a></h2>
<p><a href="/about">Find out what makes me tick.</a></p>
```

En HTML5, vous pouvez envelopper plusieurs éléments dans un seul élément `a` :

```
<a href="/about">
  <h2>About me</h2>
  <p>Find out what makes me tick.</p>
</a>
```

La seule chose qu'on ne peut pas faire, c'est insérer un élément `a` dans un autre élément `a`.

Cela pourrait sembler être un changement radical, mais les navigateurs n'auront pas grand chose à faire pour supporter ce nouveau modèle d'hyperlien. Ils le supportent déjà, même si ce type de balise n'était pas techniquement légal jusqu'à présent.

Cela paraît légèrement contre-intuitif : les navigateurs ne devraient-ils pas implémenter une spécification *existante* ? Au contraire, la nouvelle spécification documente ce que les navigateurs font déjà.

DE NOUVEAUX JOUJOUX : LES API JAVASCRIPT

Pour la documentation de CSS, il y a les spécifications CSS ; pour la documentation de l'HTML, les spécifications HTML. Mais où se trouve donc la documentation des API JavaScript comme `document.write`, `innerHTML` et `window.history` ? La

spécification JavaScript ne traite que du langage de programmation – vous n'y trouverez rien sur les API.

Jusqu'à présent, les navigateurs créaient et implémentaient les API JavaScript de manière indépendante, tout en regardant par-dessus l'épaule des concurrents. L'HTML5 va documenter ces API une bonne fois pour toutes, ce qui devrait garantir une meilleure compatibilité.

Il peut paraître bizarre d'inclure la documentation JavaScript dans la spécification d'un langage de balisage, mais rappelez-vous que l'HTML5 est né de Web Apps 1.0. JavaScript est un composant indispensable des applications web.

Des sections entières de la spécification HTML5 sont consacrées aux nouvelles API pour la création d'applications web. On y trouve `Undo-Manager`, qui permet au navigateur de suivre les changements apportés à un document. Il y a une section sur la création d'applications web hors-ligne à l'aide d'une mémoire cache. Le glisser-déposer y est décrit en détail.

Comme toujours, s'il existe déjà une implémentation, la spécification construira sur ses bases plutôt que de réinventer la roue. L'Internet Explorer de Microsoft utilise une API de glisser-déposer depuis des années, qui a donc servi de base pour le glisser-déposer de l'HTML5. Malheureusement, l'API de Microsoft est, pour rester poli, problématique. Il peut être bon de réinventer la roue si celle-ci est carrée…

Les API de l'HTML5 sont très puissantes. D'ailleurs, elles me passent complètement au-dessus de la tête. Je laisserai donc les développeurs plus intelligents que moi détailler le sujet : ces API méritent un livre à elles seules.

En attendant, il reste plein de trucs nouveaux dans l'HTML5 avec lesquels nous autres web designers allons pouvoir nous amuser. La fête commence au tout prochain chapitre.

3 LES MÉDIAS RICHES

L'histoire du Web est ponctuée d'améliorations technologiques. L'un des tout premiers ajouts à l'HTML fut l'élément `img`, qui a fondamentalement tranformé le Web. L'introduction du JavaScript permit ensuite au Web de devenir un environnement plus dynamique. Par la suite, la prolifération d'Ajax fit du Web une option viable pour créer de véritables applications.

Les normes du Web ont tellement évolué qu'il est maintenant possible de construire pratiquement n'importe quoi à l'aide de l'HTML, de CSS et de JavaScript. *Pratiquement*.

Quelques couleurs manquent à la palette des normes du Web. Si vous souhaitez publier du texte et des images, l'HTML et CSS feront l'affaire. Toutefois, si vous voulez publier de l'audio ou de la vidéo, il vous faut utiliser un plug-in comme Flash ou Silverlight.

Ces « plug-ins », ou modules d'extension, aident à combler les failles du Web. Ils permettent de publier des jeux, des films et

de la musique en ligne, assez facilement. Mais ces technologies ne sont pas ouvertes. Elles ne sont pas créées par la communauté. Elles sont sous le contrôle d'entreprises indépendantes.

Flash est une technologie puissante, mais l'utiliser revient parfois à pactiser avec le diable. Nous gagnons la possibilité de publier des médias riches sur le Web, mais nous perdons ainsi une partie de notre indépendance.

L'HTML5 est en train de combler ces lacunes. Ce faisant, il est en compétition directe avec des technologies propriétaires comme Flash et Silverlight. Sauf qu'en HTML5, les médias riches sont interprétés de façon native par le navigateur, sans l'aide d'un plug-in.

CANVAS

Quand le navigateur Mosaic a apporté la possibilité d'intégrer des images sur des pages web, il a donné un nouvel essor au Web, mais ces images sont depuis restées statiques. On peut certes créer des gif animés ; on peut utiliser JavaScript pour changer le style d'une image ; on peut générer une image de manière dynamique sur le serveur, mais dès lors qu'une image a été chargée par un navigateur, son contenu ne peut plus être modifié.

L'élément `canvas` constitue un environnement permettant de créer des images dynamiques.

L'élément lui-même est très simple. Il suffit d'en spécifier les dimensions dans la balise d'ouverture :

```
<canvas id="mon-premier-canvas" width="360" height="240">
</canvas>
```

Si vous mettez quoi que ce soit entre les balises d'ouverture et de fermeture, seuls les navigateurs ne supportant pas l'élément `canvas` le verront (FIG 3.01) :

```
<canvas id="mon-premier-canvas" width="360" height="240">
  <p>Votre navigateur ne supporte pas l'élément canvas ? »
  Voici une image classique :</p>
  <img src="chiot.jpg" alt="un mignon petit chien">
</canvas>
```

FIG 3.01 : Les utilisateurs dont le navigateur ne supporte pas l'élément canvas verront l'image d'un mignon petit chien-chien.

Le gros du travail se fait en JavaScript. Tout d'abord, vous devez référencer l'élément `canvas` et son contexte. Ici, le mot « contexte » désigne simplement API. Pour le moment, le seul contexte disponible est bidimensionnel :

```
var canvas = document.getElementById('mon-premier-canvas');
var context = canvas.getContext('2d');
```

Vous pouvez maintenant commencer à dessiner sur la surface bidimensionnelle de l'élément `canvas` à l'aide de l'API documentée dans la spécification HTML5 à cette adresse : http://bkaprt.com/html5/1[1].

L'API 2D offre grosso modo les mêmes outils que les programmes de création graphique comme Illustrator : traits, remplissage,

1. Adresse complète : http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html

ombres, formes et courbes de Bézier. La différence réside dans le fait qu'au lieu d'utiliser une interface graphique, tout doit être spécifié à l'aide de JavaScript.

Dessiner avec des lignes de code : une partie de plaisir

Voici comment spécifier que le trait doit être de couleur rouge :

```
context.strokeStyle = '#990000' ;
```

Maintenant, tout ce que vous dessinerez aura un contour rouge. Par exemple, voici la syntaxe pour dessiner un rectangle :

```
strokeRect ( gauche, haut, largeur, hauteur )
```

Si vous voulez dessiner un rectangle de 100 par 50 pixels, placé à 20 pixels de la gauche et à 30 pixels du haut de l'élément `canvas`, vous écrirez (**FIG 3.02**) :

```
context.strokeRect(20,30,100,50) ;
```

FIG 3.02 : Un rectangle, dessiné avec l'élément canvas.

C'est un exemple très simple. L'API 2D offre beaucoup d'autres méthodes : `fillStyle`, `fillRect`, `lineWidth`, `shadowColor` et bien d'autres encore.

En théorie, une image qui peut être créée dans un programme comme Illustrator peut être créée avec l'élément `canvas`. Dans la pratique, ce serait extrêmement laborieux et donnerait un code JavaScript excessivement long. Et d'ailleurs, ce n'est pas vraiment l'objet de cet élément.

Canvas, pour quoi faire ?

C'est très bien de pouvoir utiliser JavaScript et l'élément canvas pour créer des images à la volée, mais à moins d'être un irréductible masochiste, à quoi bon ?

Le vrai pouvoir de l'élément canvas, c'est que son contenu peut être modifié et redessiné à tout moment, selon les actions de l'utilisateur. Cette possibilité permet de créer des outils et des jeux qui auraient auparavant requis un plug-in comme Flash.

L'une des premières démonstrations de force de l'élément canvas fut l'œuvre de Mozilla Labs. L'application Bespin (https://bespin.mozilla.com) est un éditeur de code fonctionnant directement dans le navigateur (FIG 3.03).

C'est une application très puissante, très impressionnante. C'est aussi l'exemple type de ce qu'il ne faut *pas* faire avec l'élément canvas.

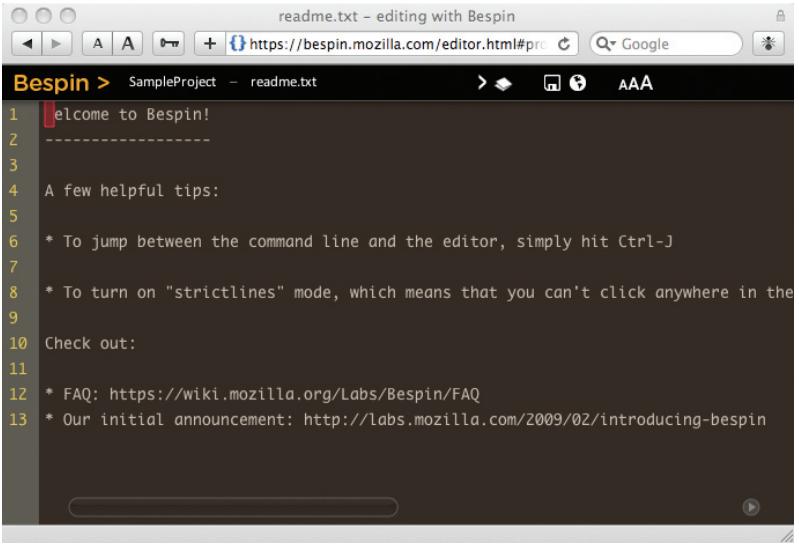


FIG 3.03 : L'application Bespin, construite avec l'élément canvas.

Accès refusé

Un éditeur de code, par sa nature, manie du texte. L'éditeur de code Bespin manie le texte contenu dans un élément `canvas`, sauf que ce n'est plus vraiment du texte, mais une suite de formes qui ressemble à du texte.

Sur le Web, chaque document peut être décrit par un *Document Object Model* (DOM). Ce DOM peut comporter de nombreux nœuds différents, les plus importants étant les éléments, le texte et les attributs. Ces trois blocs de construction suffisent à assembler à peu près n'importe quel document. L'élément `canvas` n'a pas de DOM, le contenu dessiné ne peut donc pas être représenté sous la forme d'une arborescence de nœuds.

Les lecteurs d'écran et d'autres technologies d'accessibilité dépendent de l'accès à un DOM pour interpréter un document. Pas de DOM, pas d'accès.

Le manque d'accessibilité de l'élément `canvas` est un gros problème pour l'HTML5. Heureusement, quelques personnes avisées ont formé une *task force* pour trouver des solutions (http://bkaprt.com/html5/2)[1].

L'accessibilité de l'élément `canvas` est une question importante, et je ne voudrais pas d'une solution bâclée. En même temps, je ne veux pas non plus que cet élément retarde le reste de la spécification HTML5.

Canvas futé

Tant que ce manque d'accessibilité n'est pas réglé, on peut penser que les web designers doivent s'interdire d'utiliser l'élément `canvas`. Ce n'est pas forcément le cas.

Chaque fois que j'utilise JavaScript sur un site web, c'est en tant qu'amélioration. Les visiteurs n'utilisant pas JavaScript

1. Adresse complète : http://www.w3.org/WAI/PF/html-task-force

ont quand même accès à tout le contenu, même si leur expérience n'est pas aussi dynamique que dans un environnement JavaScript. Cette approche à plusieurs niveaux, appelée JavaScript discret, peut également s'appliquer à `canvas`. Plutôt que de l'utiliser pour créer du contenu, on l'utilise pour recycler le contenu existant.

Supposons que vous ayez un tableau rempli de données, et que vous souhaitiez illustrer les tendances de ces données par un graphique. Si les données sont statiques, vous pouvez générer un graphique, par exemple avec l'API Google Chart. En revanche, si les données changent en réaction à des évènements déclenchés par l'utilisateur, `canvas` est l'outil idéal pour générer un graphique dynamique. En plus, le contenu représenté dans l'élément `canvas` est déjà accessible dans l'élément `table` préexistant.

Les grosses têtes du Filament Group ont mis au point le plug-in jQuery pour cette situation particulière (FIG 3.04 ; http://bkaprt.com/html5/3[1]).

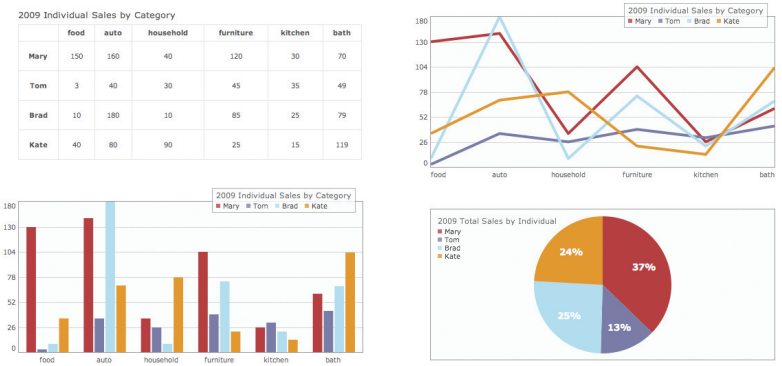


FIG 3.04 : Utilisation de l'élément canvas pour générer un graphique à partir de données entrées par l'utilisateur.

1. Adresse complète : http://www.filamentgroup.com/lab/update_to_jquery_visualize_accessible_charts_with_html5_from_designing_with/

Il existe une autre option. `canvas` n'est pas la seule API permettant de générer des images dynamiques. SVG, pour *Scalable Vector Graphics*, est un format XML pouvant décrire le même genre de formes que `canvas`. Puisque le XML est un format de données textuel, le contenu du SVG est théoriquement valable pour les lecteurs d'écran.

Dans les faits, le SVG n'a pas captivé l'imagination des développeurs comme `canvas`. Même si `canvas` est un petit nouveau, il bénéficie déjà d'une large compatibilité. Safari, Firefox, Opera et Chrome le supportent déjà. Il y a même une librairie JavaScript qui permet à Internet Explorer de l'utiliser (http://bkaprt.com/html5/4[1]).

Vu les mantras « paver le sentier des vaches » et « ne pas réinventer la roue », on pourrait trouver bizarre que le WHATWG recommande l'élément `canvas` pour l'HTML5 alors que le SVG existe déjà. Comme souvent, la spécification HTML5 ne fait que documenter ce que les navigateurs font déjà. L'élément `canvas` n'a pas été créé pour l'HTML5 ; il a été créé par Apple et implémenté dans Safari. D'autres créateurs de navigateurs ont vu ce qu'Apple faisait, l'ont aimé, et recopié.

Cela semble peu méthodique, mais c'est de là que viennent la plupart des normes du Web. Microsoft, par exemple, a créé l'objet `XMLHttpRequest` pour Internet Explorer 5 à la fin du XX^{e} siècle. Dix ans plus tard, tous les navigateurs supportent cette fonctionnalité, qui fait maintenant l'objet d'un brouillon de travail en dernier appel au W3C.

Dans le monde darwinien des navigateurs web, `canvas` se reproduit un peu partout. S'il parvient à évoluer vers plus d'accessibilité, sa survie est assurée.

1. Adresse complète : http://code.google.com/p/explorercanvas/

AUDIO

Le tout premier site que j'ai créé était une vitrine pour mon groupe de musique. Je voulais que les visiteurs du site puissent écouter les chansons du groupe. J'ai donc étudié les nombreux formats et lecteurs qui se disputaient mon attention : QuickTime, Windows Media Player, Real Audio. J'ai passé beaucoup trop de temps à me soucier des parts de marché relatives et de la compatibilité entre plateformes.

Entre-temps, le format MP3 a gagné la bataille de l'ubiquité. Mais pour offrir à nos visiteurs une façon simple d'écouter un fichier sonore, il faut tout de même utiliser une technologie propriétaire. Le lecteur Flash a gagné cette bataille-là.

Et maintenant, voilà que l'HTML5 monte sur le ring pour tenter de détrôner le champion en titre.

Il est extrêmement simple d'insérer un fichier audio dans un document HTML5 :

```
<audio src="witchitalineman.mp3">
</audio>
```

C'est un peu trop simple. Vous voudrez probablement être plus spécifique quant au comportement du fichier sonore.

Imaginons qu'il existe un immonde salaud qui déteste le Web et tous ceux qu'on y croise. Il se fiche probablement de savoir qu'il est incroyablement grossier et stupide d'insérer un fichier audio qui démarre automatiquement. Grâce à l'attribut `autoplay`, de tels actes de malveillance sont possibles :

```
<audio src="witchitalineman.mp3" autoplay>
</audio>
```

Si jamais vous utilisez l'attribut `autoplay` de cette façon, je devrai vous abattre.

Remarquez que l'attribut `autoplay` ne prend pas de valeur. C'est un attribut booléen, du nom du grand mathématicien de Cork, George Boole.

La logique informatique est entièrement basée sur la logique booléenne : un courant électrique passe, ou ne passe pas ; une valeur binaire est égale à un ou à zéro ; le résultat d'une comparaison est « vrai » ou « faux ».

Ne confondez pas les *attributs* booléens et les *valeurs* booléennes. Il serait logique de penser qu'un attribut booléen prend les valeurs « true » ou « false ». En fait, c'est l'existence même de l'attribut qui est booléenne par nature : soit l'attribut est inclus, soit il ne l'est pas. Écrire `autoplay="false"` ou `autoplay="non merci"` revient à écrire `autoplay`.

Si vous affectionnez la syntaxe du XHTML, vous pouvez écrire `autoplay="autoplay"`. Solution offerte par le Service de la répétition redondante.

Si l'idée d'imposer votre fichier audio au démarrage ne vous paraît pas assez diabolique, vous pouvez infliger encore plus de souffrance en le jouant en boucle. Un autre attribut booléen, appelé `loop`, tient ce rôle perfide :

```
<audio src="witchitalineman.mp3" autoplay loop>
</audio>
```

Si vous utilisez l'attribut `loop` avec l'attribut `autoplay`, je devrai vous abattre deux fois.

Hors de contrôle

L'élément `audio` peut représenter une bénédiction comme une malédiction. Il peut être judicieux de laisser à l'utilisateur le contrôle de la lecture du fichier audio. C'est ce qu'on fait avec l'attribut booléen `controls` :

```
<audio src="witchitalineman.mp3" controls>
</audio>
```

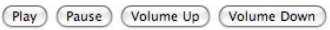
La présence de l'attribut `controls` invite le navigateur à afficher des contrôles natifs pour lire et mettre l'audio en pause, ainsi qu'ajuster le volume (FIG 3.05).

FIG 3.05 : Utilisez `controls` pour afficher les contrôles de lecture et du volume de votre fichier audio.

Si les contrôles natifs du navigateur ne vous conviennent pas, vous pouvez créer les vôtres. Avec JavaScript, vous pouvez interagir avec l'API `Audio`, qui vous donne accès à des méthodes comme `play` et `pause`, et à des propriétés comme `volume`. Voici un exemple vite fait mal fait avec des éléments `button` et d'horribles gestionnaires d'événement (FIG 3.06) :

```
<audio id="player" src="witchitalineman.mp3">
</audio>
<div>
  <button »
  onclick="document.getElementById('player').play()"> »
  Play
  </button>
  <button »
  onclick="document.getElementById('player').pause()"> »
  Pause
  </button>
  <button »
  onclick="document.getElementById('player').volume »
  += 0.1">
  Volume Up
  </button>
  <button »
  onclick="document.getElementById('player').volume »
  -= 0.1">
  Volume Down
  </button>
</div>
```

FIG 3.06 : Les contrôles obtenus avec les éléments `button`.



Mise en mémoire tampon

À un stade, la spécification HTML5 comprenait un autre attribut booléen pour l'élément `audio`. L'attribut `autobuffer` était plus poli et prévenant que l'affreux `autoplay`. Il permettait aux auteurs de dire aux navigateurs que, même si le fichier audio ne devait pas démarrer automatiquement, il allait sans doute être lu à un moment ou à un autre, et que le navigateur devait donc commencer à le charger en arrière-plan.

Cet attribut aurait pu être utile, mais Safari a malheureusement dépassé les bornes, décidant de charger les fichiers audio sans se préoccuper de la présence de l'attribut `autobuffer`. Souvenez-vous que, comme `autobuffer` était un attribut booléen, il était impossible de demander à Safari de ne pas précharger l'audio : `autobuffer="false"` revenait à écrire `autobuffer="true"` ou n'importe quelle autre valeur (http://bkaprt.com/html5/5[1]).

L'attribut `autobuffer` a maintenant été remplacé par l'attribut `preload`. Ce n'est pas un attribut booléen. Il peut prendre trois valeurs différentes : `none`, `auto` et `metadata`. Avec `preload="none",` vous pouvez maintenant demander explicitement aux navigateurs de ne pas précharger l'audio :

```
<audio src="witchitalineman.mp3" controls preload="none">
</audio>
```

Si vous n'avez qu'un seul élément `audio` sur une page, vous pouvez utiliser `preload="auto"`, mais plus il y aura d'éléments `audio` en préchargement, plus la bande passante de vos visiteurs s'en trouvera saturée.

Joue comme ci, joue comme ça

L'élément `audio` semble quasiment parfait. Il doit bien y avoir une complication quelque part ? Eh oui.

1. Adresse complète : https://bugs.webkit.org/show_bug.cgi?id=25267

Le problème de l'élément `audio` ne se trouve pas dans la spécification. Le problème vient des formats audio.

Bien que le format MP3 soit devenu omniprésent, ce n'est pas un format ouvert. Le format étant breveté, les technologies ne peuvent décoder les fichiers MP3 sans en payer les droits. Sans problème pour les grandes entreprises comme Apple ou Adobe, plus difficilement pour les plus petites ou les groupes open-source. Ainsi, Safari jouera volontiers des fichiers MP3, au contraire de Firefox.

Il existe d'autres formats audio. Le codec Vorbis (généralement sous la forme d'un fichier `.ogg`) n'est perclus d'aucun brevet. Firefox supporte le format Ogg Vorbis, mais... pas Safari.

Heureusement, il existe une autre façon d'utiliser l'élément `audio` sans avoir à faire un choix cornélien entre deux formats. Plutôt que d'utiliser l'attribut `src` dans la balise `<audio>` d'ouverture, vous pouvez spécifier plusieurs formats à l'aide de l'élément `source` :

```
<audio controls>
  <source src="witchitalineman.ogg">
  <source src="witchitalineman.mp3">
</audio>
```

Un navigateur pouvant lire les fichiers Ogg Vorbis s'arrêtera au premier élément `source`. Un navigateur pouvant lire les fichiers MP3, mais pas les fichiers Ogg Vorbis, sautera le premier élément `source` et jouera le fichier du deuxième élément `source`.

Vous pouvez aider les navigateurs en précisant les types MIME de chaque fichier source :

```
<audio controls>
  <source src="witchitalineman.ogg" type="audio/ogg">
  <source src="witchitalineman.mp3" type="audio/mpeg">
</audio>
```

L'élément `source` est un élément autonome ; par conséquent, si vous utilisez la syntaxe XHTML, assurez-vous d'inclure une barre oblique à la fin de chaque balise `<source />`.

Solution de secours

La possibilité de spécifier de multiples éléments `source` est très utile, mais certains navigateurs ne supportent pas du tout l'élément `audio` à l'heure actuelle. Vous avez deviné de quel navigateur je parlais ?

Internet Explorer et ses semblables doivent être nourris à la petite cuillère, à l'ancienne, *via* Flash. Le modèle de contenu de l'élément `audio` le permet. Tout ce qui est situé entre les balises `<audio>` et qui n'est pas un élément `source` sera exposé aux navigateurs qui ne comprennent pas l'élément `audio` :

```
<audio controls>
  <source src="witchitalineman.ogg" type="audio/ogg">
  <source src="witchitalineman.mp3" type="audio/mpeg">
  <object type="application/x-shockwave-flash" »
  data="player.swf?soundFile=witchitalineman.mp3">
    <param name="movie" »
    value="player.swf?soundFile=witchitalineman.mp3">
  </object>
</audio>
```

Dans cet exemple, l'élément `object` ne sera visible qu'aux navigateurs ne supportant pas l'élément `audio`.

On peut même aller plus loin. L'élément `object` vous permet également d'inclure une solution de secours. Cela signifie que vous pouvez toujours fournir un bon vieil hyperlien en dernier recours :

```
<audio controls>
  <source src="witchitalineman.ogg" type="audio/ogg">
  <source src="witchitalineman.mp3" type="audio/mpeg">
  <object type="application/x-shockwave-flash" »
```

```
    data="player.swf?soundFile=witchitalineman.mp3">
      <param name="movie" »
      value="player.swf?soundFile=witchitalineman.mp3">
      <a href="witchitalineman.mp3">Télécharger la chanson</
a>
    </object>
</audio>
```

Cet exemple comprend quatre niveaux de dégradation progressive :

- Le navigateur supporte l'élément `audio` et le format Ogg Vorbis.
- Le navigateur supporte l'élément `audio` et le format MP3.
- Le navigateur ne supporte pas l'élément `audio` mais le plug-in Flash est installé.
- Le navigateur ne supporte pas l'élément `audio` et le plug-in Flash est absent.

Accès complet

Le modèle de contenu de l'élément `audio` est très utile pour fournir un contenu de secours. Un contenu de secours est différent d'un contenu d'accessibilité.

Supposons qu'une transcription accompagne le fichier audio. Voici la marche à ne *pas* suivre :

```
<audio controls>
  <source src="witchitalineman.ogg" type="audio/ogg">
  <source src="witchitalineman.mp3" type="audio/mpeg">
  <p>I am a lineman for the county...</p>
</audio>
```

La transcription ne sera visible que pour les navigateurs ne supportant pas l'élément `audio`. Cette solution ne risque pas d'aider un utilisateur sourd doté d'un bon navigateur. D'ailleurs, ledit contenu d'accessibilité est souvent très utile pour tout le monde, alors pourquoi le cacher ?

```
<audio controls>
  <source src="witchitalineman.ogg" type="audio/ogg">
  <source src="witchitalineman.mp3" type="audio/mpeg">
</audio>
<p>I am a lineman for the county...</p>
```

VIDÉO

Si l'avènement du support natif des fichiers audio dans les navigateurs est enthousiasmant, le support natif de la vidéo fait saliver les web designers d'avance. Grâce à l'augmentation de la bande passante, la vidéo est de plus en plus répandue. Le plug-in Flash est pour le moment la technologie de choix pour afficher des vidéos sur le Web, mais l'HTML5 pourrait bien changer la donne.

L'élément `video` fonctionne tout comme l'élément `audio`. Il comprend les attributs optionnels `autoplay`, `loop` et `preload`. On peut spécifier l'emplacement de la vidéo avec l'attribut `src` de l'élément `video`, ou en plaçant des éléments `source` entre les balises `<video>`. Vous pouvez laisser au navigateur le soin de fournir une interface utilisateur avec l'attribut `controls`, ou vous pouvez scripter vos propres contrôles.

La principale différence entre le contenu audio et le contenu vidéo, c'est que les films, par nature, prennent plus de place sur l'écran. Vous voudrez donc sans doute préciser des dimensions :

```
<video src="film.mp4" controls width="360" height="240">
</video>
```

Vous pouvez choisir une image représentative de la vidéo et demander au navigateur de l'afficher à l'aide de l'attribut `poster` (FIG 3.07) :

```
<video src="film.mp4" controls width="360" »
height="240" poster="apercu.jpg">
</video>
```

FIG 3.07 : Cet aperçu est affiché à l'aide de l'attribut `poster`.

La guerre des formats vidéo concurrents est encore plus sanglante que celle des formats audio. Parmi les acteurs principaux, citons MP4, qui est breveté, et Theora Video, qui ne l'est pas. Une fois de plus, vous devrez proposer des encodages alternatifs et un contenu de secours :

```
<video controls width="360" height="240" »
poster="apercu.jpg">
  <source src="film.ogv" type="video/ogg">
  <source src="film.mp4" type="video/mp4">
  <object type="application/x-shockwave-flash" »
  width="360" height="240" »
  data="player.swf?file=film.mp4">
    <param name="movie" »
    value="player.swf?file=film.mp4">
    <a href="film.mp4">Télécharger le film</a>
  </object>
</video>
```

Les auteurs de la spécification HTML5 espéraient à l'origine spécifier un niveau de référence pour le support des formats. Hélas, les créateurs de navigateurs ne pouvaient s'accorder sur un seul format.

L'avènement du natif

La possibilité d'intégrer des vidéos de façon native dans des pages web pourrait bien être l'ajout le plus enivrant depuis

l'introduction de l'élément `img`. Les gros pontes comme Google n'ont pas manqué d'exprimer leur enthousiasme. Vous pouvez avoir un aperçu de ce qu'ils ont prévu pour YouTube à l'adresse http://youtube.com/HTML5.

L'un des problèmes avec l'utilisation de plug-ins pour les médias riches, c'est que le contenu du plug-in est isolé du reste du document, alors que des médias riches natifs s'accommodent très bien d'autres technologies, comme JavaScript et CSS.

L'élément `video` n'est pas seulement scriptable, il est également « stylable » (FIG 3.08).

FIG 3.08 : Style appliqué à un élément `video`.

Essayez donc de faire ça avec un plug-in...

L'audio et la vidéo sont les bienvenus dans l'HTML5, mais le Web n'est pas qu'un support de diffusion. Il est interactif. Les formulaires sont la façon la plus ancienne, et la plus puissante, de permettre cette interaction. Au chapitre 4, nous allons nous intéresser à la nouvelle mouture des formulaires en HTML5.

4 WEB FORMS 2.0

Quand Javascript fut introduit dans les navigateurs web, il fut immédiatement utilisé pour deux tâches : les images réactives et les formulaires améliorés. Quand CSS arriva avec sa pseudo-classe `:hover`, les web designers n'eurent plus besoin d'utiliser JavaScript pour obtenir un simple effet au survol de l'image.

C'est une tendance récurrente. Si un modèle est suffisamment populaire, il évoluera presque certainement d'une solution scriptée vers quelque chose de plus déclaratif. C'est pour cette raison que CSS3 comprend encore plus d'animations qui requéraient auparavant l'utilisation de JavaScript.

Pour ce qui est des formulaires améliorés, CSS est limité. C'est là que l'HTML5 entre en jeu. Suivant le même modèle migratoire (solution scriptée vers solution déclarative), la spécification apporte de nombreuses améliorations.

Ces fonctionnalités faisaient à l'origine partie d'une spécification du WHATWG appelé Web Forms 2.0, fondée sur les travaux existants du W3C. Cette spécification fait désormais partie de l'HTML5.

PLACEHOLDER

Voici un modèle de script DOM répandu, souvent utilisé pour les formulaires de recherche :

1. Quand un champ du formulaire est vide, insérer un texte de substitution.
2. Quand l'utilisateur sélectionne ce champ, retirer le texte de substitution.
3. Si l'utilisateur quitte le champ en le laissant vide, rétablir le texte de substitution.

Le texte de substitution est généralement affiché d'une couleur plus claire qu'une véritable valeur, à l'aide de CSS, de JavaScript, ou d'une combinaison des deux.

Dans un document HTML5, vous pouvez simplement utiliser l'attribut `placeholder` (FIG 4.01) :

```
<label for="hobbies">Vos hobbies</label>
<input id="hobbies" name="hobbies" type="text" »
placeholder="Tournage de pouces">
```

L'attribut `placeholder` fait des merveilles avec les navigateurs qui le supportent, mais hélas, ils sont encore bien peu nombreux. C'est à vous de décider de la manière d'aborder les navigateurs incompatibles.

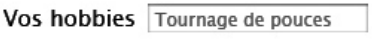


FIG 4.01 : « Tournage de pouces » est affiché dans le champ via l'attribut `placeholder`.

Vous pouvez décider de ne rien faire du tout. Après tout, c'est une fonctionnalité bien sympathique, mais pas indispensable. Vous pouvez aussi vous rabattre sur une solution en JavaScript. Dans ce cas, vous devrez vous assurer que la solution en question ne s'applique qu'aux navigateurs qui ne comprennent pas l'attribut `placeholder`.

Voici une petite fonction JavaScript générique qui permet de vérifier si un élément supporte un attribut particulier :

```
function elementSupporteAttribut(element,attribut) {
  var test = document.createElement(element);
  if (attribut in test) {
    return true;
  } else {
    return false;
  }
}
```

Cette fonction crée un élément « fantôme » dans la mémoire (mais pas dans votre document), puis vérifie si ce prototype d'élément comprend une propriété portant le même nom que l'attribut dont vous testez la présence. La fonction renvoie `true` ou `false`.

Avec cette fonction, vous pouvez vous assurer que la solution JavaScript ne s'applique qu'aux navigateurs qui ne supportent pas l'attribut `placeholder` :

```
if (!elementSupporteAttribut('input','placeholder')) {
  // Placez ici la solution de secours en JavaScript.
}
```

AUTOFOCUS

« Salut ! Je suis le modèle autofocus. Vous vous souvenez peut-être de ma prestation dans "Google : J'ai de la chance" et "Twitter : Quoi de neuf ?" »

Il s'agit d'un modèle simple comprenant une seule étape, facile à programmer en JavaScript :

1. Quand le document est chargé, sélectionner automatiquement un champ précis d'un formulaire.

L'HTML5 vous permet de faire cela à l'aide de l'attribut `autofocus` :

```
<label for="statut">Quoi de neuf ?</label>
<input id="statut" name="statut" type="text" autofocus>
```

Le seul problème avec ce modèle, c'est qu'il peut être franchement casse-pieds. Quand je surfe sur le Web, j'utilise souvent la barre espace pour faire défiler une page, comme on déroulerait un journal. Au lieu de cela, sur les sites comme Twitter qui utilisent le modèle autofocus, je me retrouve à remplir d'espaces un champ de formulaire.

Je vois bien pourquoi l'attribut `autofocus` a été ajouté à l'HTML5 (le sentier des vaches…), mais j'ai peur qu'il ne soit pas vraiment pratique, en script comme en natif. C'est une fonctionnalité qui peut s'avérer tout aussi utile qu'exaspérante. Prenez bien le temps de réfléchir avant de l'implémenter.

Il y a au moins un avantage à ce que ce modèle passe du script à la balise. En théorie, les utilisateurs pourront désactiver l'option dans les préférences de leur navigateur. En pratique, aucun navigateur ne le permet encore, mais le modèle est encore jeune. Pour le moment, la seule façon de désactiver l'autofocus consiste à désactiver entièrement JavaScript. C'est un peu comme s'arracher les yeux pour se protéger d'une lumière vive, mais ça marche.

Comme avec l'attribut `placeholder`, vous pouvez vérifier le support de l'attribut `autofocus` et proposer une solution scriptée de secours :

```
if (!elementSupporteAttribut('input','autofocus')){
  document.getElementById('statut').focus();
}
```

L'attribut `autofocus` ne marche pas seulement avec l'élément `input`. Il peut est utilisé avec n'importe quel type de champ, comme `textarea` ou `select`, mais une fois seulement par document.

REQUIRED

L'une des utilisations les plus courantes de JavaScript est la validation de formulaire côté client. Une fois encore, l'HTML5 migre la solution scriptée vers une solution balisée. Ajoutez simplement l'attribut booléen `required` :

```
<label for="pass">Votre mot de passe</label>
<input id="pass" name="pass" type="password" required>
```

Théoriquement, cela permet aux navigateurs d'empêcher la validation du formulaire si les champs requis n'ont pas été remplis. Bien que les navigateurs ne prennent pas encore l'attribut `required` en charge, vous pouvez quand même l'utiliser dans votre validation de formulaire en JavaScript. Au lieu de garder une liste de tous les champs requis dans votre script ou d'ajouter `class="required"` à vos balises, vous pouvez désormais vérifier la présence de l'attribut `required`.

AUTOCOMPLETE

Les navigateurs ne se contentent pas d'afficher des pages web. La plupart des navigateurs offrent des fonctionnalités supplémentaires conçues pour rendre la navigation plus fonctionnelle, plus sûre ou plus pratique. En général, elles sont très utiles, mais elles peuvent parfois être désagréables, voire carrément dangereuses. Je veux bien que mon navigateur conserve mes coordonnées, mais je préfère éviter qu'il conserve mes identifiants bancaires, au cas où quelqu'un déciderait de voler mon ordinateur.

L'HTML5 vous permet de désactiver l'autocomplétion champ par champ. L'attribut `autocomplete` n'est pas booléen, mais il ne peut pourtant prendre que deux valeurs, « on » et « off » :

```
<form action="/autodestruction" autocomplete="off">
```

Par défaut, les navigateurs supposeront que la valeur de l'attribut `autocomplete` est « on », leur permettant de préremplir les formulaires.

Vous pouvez avoir le beurre et l'argent du beurre. Si vous voulez autoriser le préremplissage d'un formulaire, mais le désactiver pour un ou plusieurs champs, c'est possible :

```
<input type="text" name="usageunique" »
autocomplete="off">
```

Il n'existe pas de solution de secours en JavaScript pour les navigateurs qui ne supportent pas l'attribut `autocomplete`. Dans ce cas, le nouvel attribut HTML5 complète le comportement existant des navigateurs plutôt que de remplacer une solution scriptée.

La possibilité de désactiver l'autocomplétion dans les navigateurs peut sembler être un drôle d'ajout à la spécification HTML5. L'HTML5 est censé codifier les modèles répandus, et ce n'est pas un cas très coutumier. Mais étant donné les risques de sécurité potentiels induits par l'autocomplétion, il paraît logique de permettre aux propriétaires de sites web de neutraliser cette fonctionnalité particulière.

DATALIST

Le nouvel élément `datalist` vous permet d'hybrider un élément `input` classique avec un élément `select`. Grâce à l'attribut `list`, vous pouvez combiner une liste d'options et une zone de saisie (FIG 4.02) :

```
<label for="maplanete">Votre planète de naissance</label>
<input type="text" name="maplanete" id="maplanete" »
list="planetes">
<datalist id="planetes">
  <option value="Mercure">
  <option value="Vénus">
  <option value="Terre">
```

```
    <option value="Mars">
    <option value="Jupiter">
    <option value="Saturne">
    <option value="Uranus">
    <option value="Neptune">
  </datalist>
```

Cela permet aux utilisateurs de sélectionner une option proposée dans la liste ou d'entrer une valeur qui n'est pas la liste du tout. Cela s'avère très pratique dans les situations où l'on ajouterait normalement un champ du genre : « Si autre, veuillez préciser… » (FIG 4.03).

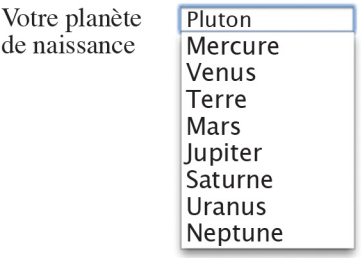


FIG 4.02 : Le nouvel élément `datalist`.

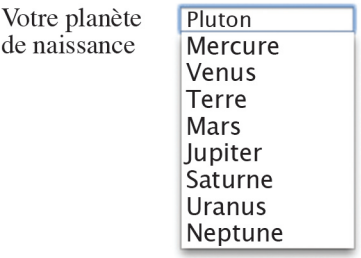


FIG 4.03 : L'élément `datalist`, avec la possibilité pour l'utilisateur d'entrer une valeur absente de la liste.

L'élément `datalist` est une amélioration appréciable et discrète pour les formulaires. Si un navigateur ne supporte pas `datalist`, le champ se comporte alors comme une zone de saisie normale.

TYPES DE CHAMPS

L'attribut `type` de l'élément `input` s'enrichit considérablement en HTML5. Il y a tellement de sentiers à paver qu'on se croirait sur un chantier à Dubaï.

Recherche

Un élément input avec l'attribut `type="search"` se comportera quasiment de la même manière qu'un élément `input` avec un attribut `type="text"` :

```
<label for="recherche">Rechercher</label>
<input id="recherche" name="recherche" type="search">
```

La seule différence entre « text » et « search », c'est qu'un navigateur pourra afficher un champ de recherche différemment pour refléter le style des champs de recherche du système d'exploitation. C'est exactement ce que fait Safari (FIG 4.04).

FIG 4.04 : Safari applique le style de Mac OS aux champs de recherche.

Coordonnées

Il existe trois nouvelles valeurs pour l'attribut `type` correspondant à différents types de coordonnées, adresse mail, sites web et numéros de téléphone :

```
<label for="adressemail">Adresse mail</label>
<input id="adressemail" name="adressemail" type="email">
<label for="siteweb">Site web</label>
<input id="siteweb" name="siteweb" type="url">
<label for="telephone">Téléphone</label>
<input id="telephone" name="telephone" type="tel">
```

Une fois encore, ces champs se comporteront comme des zones de saisie, mais les navigateurs disposeront d'un peu plus d'informations à propos des données attendues dans le champ.

Safari prétend supporter ces nouveaux types de champs, mais on ne constate à première vue aucune différence dans le navigateur de bureau par rapport à l'attribut `type="text"`. Cependant, si l'on commence à interagir avec le même formulaire sous Safari Mobile, les différences deviennent évidentes. Le navigateur affiche un clavier différent sur l'écran selon la valeur de l'attribut `type` (FIG 4.05).

FIG 4.05 : Safari Mobile affiche un clavier différent sur l'écran selon la valeur de l'attribut `type`.

Bravo WebKit, c'est finement joué.

Sliders

De nombreuses librairies JavaScript offrent des widgets préconçus à utiliser dans vos applications web. Ils marchent bien du moment que JavaScript est activé. Il serait bon que nos utilisateurs n'aient pas à télécharger un fichier JavaScript chaque fois que nous souhaitons ajouter un contrôle intéressant sur nos pages.

Le slider est un exemple classique. Jusqu'à maintenant, il nous fallait utiliser JavaScript pour émuler ce type d'élément

interactif. En HTML5, grâce à `type="range"`, les navigateurs peuvent maintenant proposer un contrôle natif :

```
<label for="montant">Combien ?</label>
<input id="montant" name="montant" type="range">
```

Actuellement, Safari et Opera supportent tous deux ce type de champ, proposant des contrôles semblables (FIG 4.06).

FIG 4.06 : Le type de champ `range` sous Safari et Opera.

Par défaut, la valeur saisie acceptera une plage allant de zéro à cent. Vous pouvez définir vos propres valeurs minimum et maximum à l'aide des attributs `min` et `max` :

```
<label for="note">Votre note</label>
<input id="note" name="note" type="range" »
min="1" max="5">
```

C'est bien beau pour les utilisateurs de Safari et d'Opera, mais les autres navigateurs afficheront un simple champ de saisie textuel. Cela peut suffire, mais vous voudrez peut-être définir un contenu de secours en JavaScript pour les navigateurs qui ne supportent pas `type="range"`.

Test

Le test du support natif des types de champs requiert une manipulation semblable au test du support des attributs. Il vous faudra une fois de plus créer un élément `input` « fantôme » dans la mémoire, puis définir l'attribut `type` que vous souhaitez tester. Si vous obtenez la valeur « text » en demandant la valeur de la propriété `type`, vous saurez que le navigateur ne supporte pas la valeur entrée.

Voici un programme d'exemple, même si vous pouvez sûrement écrire quelque chose de beaucoup plus élégant :

```
function inputSupporteType(test) {
  var input = document.createElement('input');
  input.setAttribute('type',test);
  if (input.type == 'text') {
    return false;
  } else {
    return true;
  }
}
```

Vous pouvez ensuite utiliser cette fonction pour vous assurer qu'un widget JavaScript est fourni aux navigateurs qui ne supportent pas tel ou tel type de champs :

```
if (!inputSupporteType('range')) {
  // Placez ici la solution de secours en JavaScript.
}
```

Un contrôle natif sera sûrement chargé plus vite qu'une solution scriptée, qui devra attendre la fin du chargement du DOM. De même, un contrôle natif sera généralement plus accessible qu'un contrôle scripté, même si, curieusement, le contrôle `range` de Safari n'est pour l'instant pas accessible au clavier !

Boutons fléchés

Le contrôle natif `range` n'expose pas la valeur sous-jacente à l'utilisateur. Au lieu de cela, le nombre est affiché sur la forme d'un slider. Cela convient pour certains types de données, mais il peut parfois être préférable de permettre à l'utilisateur de voir et de choisir la valeur numérique. C'est à cela que sert `type="number"` :

```
<label for="montant">Combien ?</label>
<input id="montant" name="montant" type="number" »
min="5" max="20">
```

L'utilisateur peut non seulement entrer directement une valeur dans le champ de saisie, mais également se servir des flèches pour modifier celle-ci (FIG 4.07).

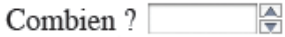


FIG 4.07 : Boutons fléchés utilisant le type `number`.

Le type de champ `number` est un hybride de `text` et de `range`. Il permet aux utilisateurs d'entrer directement des valeurs, comme dans un champ `text`, mais il permet également aux navigateurs de s'assurer que seules des valeurs numériques sont entrées, comme avec le contrôle `range`.

Date et heure

L'un des widgets JavaScript les plus prisés est le calendrier. Vous connaissez la chanson : vous réservez un avion ou vous créez un évènement et vous devez choisir une date. Un petit calendrier s'affiche à cet effet.

Ces calendriers font tous la même chose, mais vous remarquerez que leur implémentation est légèrement différente d'un site à l'autre. Un widget calendrier natif permettrait de lisser ces divergences et de réduire la charge cognitive pendant le choix de la date.

L'HTML5 introduit un tas de nouveaux types de champs spécifiquement conçus pour la date et l'heure :

- `date` renvoie l'année, le mois et le jour.
- `datetime` renvoie l'année, le mois et le jour en combinaison avec les heures, les minutes, les secondes et le fuseau horaire.
- `datetime-local` est identique mais sans le fuseau horaire.
- `time` renvoie les heures, les minutes et les secondes.
- `month` renvoie l'année et le mois sans le jour.

Tous ces types de champs utilisent le format standardisé AAAA-MM-JJThh:mm:ss.Z (où A est l'année, M le mois, J le jour, h les heures, m les minutes, s les secondes et Z le fuseau horaire).

Prenons par exemple la date et l'heure de la fin de la première guerre mondiale, 11 h 11 le 11 novembre 1918 :

- `date` : 1918-11-11
- `datetime` : 1918-11-11T11:11:00+01
- `datetime-local` : 1918-11-11T11:11:00
- `time` : 11:11:00
- `month` : 1918-11

Il n'existe pas de type de champ `year`, bien qu'il y ait un type de champ `week` qui renvoie un nombre entre 1 et 53 en combinaison avec l'année.

Il est très simple d'utiliser ces types de champs :

```
<label for="dtstart">Date de départ</label>
<input id=”dtstart" name="dtstart" type="date">
```

Opera implémente ces types de champs avec son incontournable touche disgracieuse (FIG 4.08).

FIG 4.08 : L'affichage du calendrier natif d'Opera, avec la touche moche.

Comme toujours, les navigateurs qui ne supportent pas ces types de champs se rabattront sur un champ de saisie normal. Dans cette situation, vous pouvez demander aux utilisateurs d'entrer la date et l'heure au format ISO, ou utiliser la librairie

JavaScript de votre choix afin de générer un widget. Assurez-vous de commencer par vérifier le support natif :

```
if (!inputSupporteType('date')) {
  // Placez ici un widget calendrier.
}
```

En JavaScript, même le calendrier le plus élégamment écrit nécessitera un code complexe pour générer le tableau des jours et gérer les évènements lors du choix de la date. Un calendrier natif sera considérablement plus simple et rapide, en plus d'être identique d'un site à l'autre.

Choix des couleurs

En HTML5, le plus ambitieux des remplacements de widgets est peut-être le type de champ `color`. Celui-ci utilise le fameux format hexadécimal : `#000000` pour le noir, `#FFFFFF` pour le blanc.

```
<label for="bgcolor">Couleur d'arrière-plan</label>
<input id="bgcolor" name="bgcolor" type="color">
```

L'idée étant d'amener les navigateurs à implémenter un widget natif pour le choix des couleurs, comme il en existe déjà dans toutes les autres applications de votre ordinateur. Jusqu'à présent, aucun navigateur ne l'a implémenté, mais ça promet !

En attendant, vous pouvez opter pour une solution en JavaScript, mais assurez-vous de tester le support natif, afin que votre code soit à l'épreuve du temps.

Libre-service

Tous ces nouveaux types de champs ont deux utilités : ils permettent aux navigateurs d'afficher des contrôles natifs adaptés au type de données attendu et de valider la valeur entrée. Ces ajouts à l'HTML5 couvrent la majorité des scénarios, mais vous devrez parfois valider une valeur qui ne rentre dans aucune des nouvelles catégories.

La bonne nouvelle, c'est que vous pouvez utiliser l'attribut `pattern` pour spécifier précisément le type de valeur attendu. La mauvaise nouvelle, c'est qu'il vous faudra utiliser une expression rationnelle :

```
<label for="cp">Code postal</label>
<input id="cp" name="cp" pattern="[\d]{5}">
```

En général, vous n'aurez jamais besoin d'utiliser l'attribut `pattern`. Si cela devait toutefois vous arriver, vous auriez, croyez-moi, tout mon soutien.

TOURNÉS VERS LE FUTUR

Les formulaires ont pris un coup de jeune en HTML5. Le fardeau traditionnellement porté par JavaScript passe progressivement sur les épaules des balises. Nous sommes en pleine phase de transition, et seuls quelques navigateurs supportent quelques nouvelles fonctionnalités. Nous ne pouvons pas encore mettre JavaScript au placard, mais nous ne sommes pas si loin d'un avenir meilleur.

La validation côté client va devenir beaucoup plus simple, même si vous ne devez jamais vous en contenter et valider également les valeurs sur le serveur. Vos utilisateurs n'auront plus à télécharger de librairies JavaScript pour générer des contrôles de formulaire ; tout sera géré par le navigateur de façon native.

Vous voyez sûrement les avantages à avoir des contrôles natifs pour les calendriers et les sliders, mais je parie que vous vous demandez : « Est-ce que je peux les styler ? »

C'est une bonne question. À l'heure actuelle, la réponse est « non ». Adressez-vous au groupe de travail CSS pour toute réclamation.

Cela pourrait vous rebuter. Si vous trouvez que l'implémentation standard d'un élément de formulaire est un peu sommaire, vous préfèrerez peut-être utiliser un widget JavaScript vous offrant un plus grand contrôle.

J'aimerais alors que vous vous posiez une autre question : « Est-ce que je *dois* les styler ? »

Rappelez-vous, le Web, ce n'est pas que le contrôle. Si un visiteur de votre site est habitué à tel bidule de formulaire natif, vous ne lui rendrez pas service en remplaçant celui-ci par votre propre widget, fût-il plus joli.

Personnellement, j'aimerais bien voir les créateurs de navigateurs se battre pour offrir les contrôles de formulaire HTML5 les plus beaux et les plus simples d'utilisation. C'est une guerre des navigateurs qui me siérait.

Mettons maintenant les formulaires de côté, et intéressons-nous à la nouvelle sémantique croustillante de l'HTML5.

3 LA SÉMANTIQUE

L'HTML ne nous procure pas beaucoup d'éléments avec lesquels travailler. Le choix est digne d'une épicerie de quartier, plus que d'une grande surface.

Nous avons des paragraphes, des listes et des titres, mais pas d'événements, d'informations ou de recettes. L'HTML nous donne bien un élément pour baliser les abréviations, mais rien pour les prix.

Manifestement, cette restriction n'est pas rédhibitoire. Voyez plutôt la diversité incroyable des sites web. Même si l'HTML ne prévoit pas d'élément spécifique pour baliser chaque type de contenu, il est « suffisamment » flexible.

Pour paraphraser Winston Churchill, l'HTML est le pire langage de balisage, à l'exception de tous les autres.

EXTENSIBILITÉ

D'autres langages de balisage vous permettent d'inventer l'élément que vous voulez. En XML, si vous voulez un élément `event` ou `price`, il vous suffit de prendre votre petit clavier et de le créer. L'inconvénient de cette liberté, c'est qu'il vous faut ensuite apprendre à un parseur ce que `event` et `price` veulent dire. L'avantage d'avoir un nombre d'éléments limité, c'est que chaque *user agent* connaît chaque élément. Les navigateurs ont une connaissance intégrée de l'HTML. Cela ne serait pas possible si nous pouvions créer des noms d'éléments à volonté.

L'HTML offre un joker aux web designers pour ajouter une valeur sémantique aux éléments : l'attribut `class`. Cet attribut nous permet de classer des instances spécifiques d'un élément comme classe ou type spécial de cet élément. Le fait que les navigateurs ne comprennent pas le vocabulaire utilisé dans nos attributs `class` n'affecte pas le rendu de nos documents.

Si, à ce stade, vous vous dites « attends une seconde, les classes, c'est pas pour CSS ? », eh bien vous n'avez pas tout à fait tort. Le sélecteur de classe CSS est un exemple de technologie qui utilise l'attribut `class`, mais ce n'est pas la *seule* raison d'utiliser les classes. Les classes peuvent également être utilisées dans un script DOM. Elles peuvent même être utilisées par les navigateurs si le nom des classes suit une convention entendue, comme c'est le cas avec les microformats.

Microformats

Les microformats sont un ensemble de conventions fixées par une communauté. Ces formats utilisent l'attribut `class` pour combler les lacunes les plus flagrantes de l'HTML : hCard pour les coordonnées, hCalendar pour les évènements, hAtom pour les informations. Puisqu'il y a un consensus de la communauté concernant le nom des classes à utiliser, des parseurs et des extensions de navigateurs acceptent maintenant ces modèles spécifiques.

Les microformats sont limités par essence. Ils ne prétendent pas résoudre chaque cas d'utilisation potentiel, mais se concentrent

plutôt sur les « fruits à portée de main ». Ils traitent 80 % des cas d'utilisation en produisant 20 % de l'effort. Il est relativement facile de déterminer quels sont ces fruits : il suffit de regarder le contenu déjà en place, ou en d'autres termes familiers, de paver le sentier des vaches.

Ça vous rappelle quelque chose ? Les microformats et l'HTML5 sont construits sur des philosophies très similaires. En fait, ma description des microformats (des conventions fixées par une communauté) pourrait tout aussi bien s'appliquer à l'HTML5.

Vider l'océan à la petite cuillère

Le fait d'avoir utilisé les microformats comme modèles pour le développement de l'HTML5 n'est pas au goût de tout le monde. La loi des 80/20 est assez bonne pour le monde rudimentaire des noms de classes, mais l'est-elle pour le langage de balisage le plus important du monde ?

Certaines personnes pensent que l'HTML doit être extensible à l'infini. Il ne serait donc pas suffisant de fournir des solutions pour la majorité des cas d'utilisation ; le langage devrait fournir des solutions pour tous les cas d'utilisation potentiels.

L'argument le plus éloquent en faveur d'une telle extensibilité fut peut-être celui de John Allsopp dans son brillant article sur *A List Apart*, « Semantics in HTML5 » (http://bkaprt.com/html5/6[1]) :

> *Nous n'avons pas besoin d'ajouter des termes spécifiques au vocabulaire de l'HTML, nous devons ajouter un mécanisme qui permet d'ajuster la richesse sémantique d'un document selon les besoins.*

Il existe déjà des technologies à cet effet. RDFa permet aux auteurs d'ajouter du vocabulaire personnalisé à des documents HTML. Mais à la différence des microformats, qui utilisent

1. Adresse complète : http://www.alistapart.com/articles/semanticsinHTML5

simplement un ensemble déterminé de noms de classes, RDFa utilise des espaces de nommage pour permettre la création d'une variété infinie de formats. Ainsi, là où un microformat utiliserait une balise du type `<h1 class="sommaire">`, RDFa utilisera `<h1 property="monformat:sommaire">`.

Il est évident que RDFa est potentiellement très puissant, mais son expressivité a un prix. Les espaces de nommage introduisent une couche de complexité supplémentaire qui s'accommode mal de la nature relativement simple de l'HTML.

Le débat sur les espaces de nommage n'est pas nouveau. Il y a quelques années, dans un article de son blog, Mark Nottingham songeait à leurs effets secondaires potentiellement destructeurs (http://bkaprt.com/html5/7[1]) :

> *Ce qui m'a paru intéressant avec l'extensibilité de l'HTML, c'est que les espaces de nommage n'étaient pas nécessaires. Netscape a ajouté blink, Microsoft a ajouté marquee, et ainsi de suite. Je soulignerai le fait que si l'on avait eu des espaces de nommage en HTML dès le départ, on aurait légitimé et institutionnalisé les différences entre navigateurs au lieu de converger vers la même solution.*

Contre l'extensibilité infinie, voilà une argumentation percutante en faveur d'un vocabulaire limité, fondé sur le consensus de la communauté.

L'HTML5 sera probablement accompagné d'une méthode permettant d'accroître sa sémantique native. Bien sûr, l'attribut `class` est toujours valable, et les microformats continueront donc à fonctionner comme auparavant. L'HTML5 sera peut-être altéré pour devenir compatible avec RDFa, ou utilisera peut-être son propre vocabulaire pour les « microdonnées ». Dans tous les cas, cette extensibilité aura vraisemblablement très peu d'intérêt pour la plupart des web designers. Ce qui compte vraiment, c'est la sémantique native, déterminée par une communauté et implémentée par les créateurs de navigateurs.

1. Adresse complète : http://www.mnot.net/blog/2006/04/07/extensibility

NOUVEAUX ÉLÉMENTS

L'HTML5 implémente une poignée de nouveaux éléments de type en-ligne (*inline*) venant rejoindre `span`, `strong`, `em`, `abbr` et quelques autres dans l'arsenal existant. Oh, et on ne dit plus « en-ligne ». On préfèrera dire qu'ils décrivent la « sémantique de niveau texte ».

mark

En parcourant une liste de résultats de recherche, vous verrez souvent le terme recherché mis en surbrillance dans chaque résultat. Vous pourriez marquer chaque occurrence du terme recherché à l'aide d'un élément `span`, mais `span` n'est qu'une béquille dépourvue de sens, tout juste bonne à servir de support à une classe à des fins de style.

Vous pourriez utiliser `em` ou `strong`, mais ce serait sémantiquement incorrect ; on ne veut pas donner une quelconque importance sur le terme recherché mais simplement trouver une façon de le mettre en surbrillance.

Utilisez l'élément `mark` :

```
<h1>Résultats de recherche pour 'licorne'</h1>
<ol>
  <li><a href="http://clearleft.com/">
  Chevauche la <mark>licorne</mark> UX sur »
  l'arc-en-ciel du Web.
  </a></li>
</ol>
```

L'élément `mark` n'accorde aucune importance à son contenu, sauf pour montrer que celui-ci présente temporairement un intérêt. La spécification nous dit que `mark` indique « un morceau de texte marqué ou mis en surbrillance dans un document à des fins de référence, du fait de sa pertinence dans un autre contexte ».

L'élément `mark` est utilisable dans d'autres contextes que les résultats de recherche, mais je suis bien infichu de trouver un seul exemple du genre.

time

hCalendar est l'un des microformats les plus populaires car il répond à un besoin très courant : marquer des évènements pour que les utilisateurs puissent les ajouter directement à leur calendrier.

Avec hCalendar, la seule étape délicate consiste à décrire les dates et les heures de manière lisible pour une machine. Les humains aiment donner des dates sous la forme « 25 mai » ou « mercredi prochain », mais les parseurs veulent une jolie date au format ISO : AAAA-MM-JJThh:mm:ss.

La communauté des microformats a trouvé quelques solutions intelligentes à ce problème, en utilisant par exemple l'élément `abbr` :

```
<abbr class="dtstart" title="1992-01-12">
  12 janvier 1992
</abbr>
```

Si le fait d'utiliser l'élément `abbr` de cette manière vous incommode, il existe de nombreuses autres façons de marquer des dates et des heures lisibles par une machine dans des microformats à l'aide du modèle de classe `value`. En HTML5, ce problème est réglé par le nouvel élément `time` :

```
<time class="dtstart" datetime="1992-01-12">
  12 janvier 1992
</time>
```

L'élément `time` peut être utilisé pour les dates, les heures, ou une combinaison des deux :

```
<time datetime="17:00">17 h</time>
<time datetime="2010-04-07">7 avril</time>
<time datetime="2010-04-07T17:00">17 h le 7 avril</time>
```

Vous n'êtes pas obligé de définir l'attribut `datetime`, mais si vous ne le faites pas, vous devrez alors exposer la valeur à l'utilisateur final :

```
<time>2010-04-07</time>
```

meter

L'élément `meter` peut être utilisé pour baliser des mesures, du moment que ces mesures font partie d'une échelle avec une valeur minimum et une valeur maximum.

```
<meter>9 chats sur 10</meter>
```

Vous n'avez pas à exposer la valeur maximum si vous ne le souhaitez pas. Vous pouvez utiliser l'attribut `max` à la place :

```
<meter max="10">9 chats</meter>
```

Il existe un attribut `min` correspondant. Vous pouvez également jouer avec les attributs `high`, `low` et `optimum`. Si vous le souhaitez, vous pouvez même cacher la mesure elle-même dans un attribut `value`.

```
<meter low="-273" high="100" min="12" max="30" »
optimum="21" value="25">
  Il fait plutôt chaud pour cette période de l'année.
</meter>
```

progress

Alors que `meter` est utile pour décrire quelque chose qui a déjà été mesuré, l'élément `progress` vous permet de baliser une valeur qui est en plein changement :

```
Votre profil est complet à <progress>60 %</progress>.
```

Une fois encore, vous pouvez utiliser les attributs `min`, `max` et `value` :

```
<progress min="0" max="100" value="60"></progress>
```

L'élément `progress` est particulièrement utile en combinaison avec un script DOM. Vous pouvez utiliser JavaScript pour mettre à jour la valeur en temps réel, permettant ainsi au navigateur de communiquer ce changement à l'utilisateur. Pratique pour les envois de fichiers avec Ajax.

STRUCTURE

En 2005, Google diligenta une étude pour voir quels fruits étaient à portée de main sur les sentiers battus du Web (http://code.google.com/webstats/).

Un parseur s'occupa de parcourir plus d'un milliard de pages web et répertoria les noms de classes les plus courants. Les résultats furent sans surprise. Les classes comme `header`, `footer` et `nav` tenaient le haut du pavé. Cette sémantique émergente correspond bien à certains des nouveaux éléments structurels de l'HTML5.

section

L'élément `section` est utilisé pour regrouper du contenu de manière thématique. Cela rappelle l'élément `div`, souvent utilisé comme un conteneur de contenu générique. La différence, c'est que `div` n'a aucune signification sémantique ; il ne nous dit rien de son contenu. L'élément `section`, en revanche, est utilisé explicitement pour regrouper du contenu apparenté.

Vous pourrez peut-être remplacer quelques uns de vos éléments `div` par des éléments `section`, mais demandez-vous toujours si tout le contenu est apparenté.

```
<section>
  <h1>DOM Scripting</h1>
  <p>Ce livre est destiné aux designers »
  plutôt qu'aux programmeurs.</p>
  <p>Par Jeremy Keith</p>
</section>
```

header

La spécification HTML5 décrit l'élément `header` comme un conteneur pour « un groupe d'outils d'introduction ou de navigation ». Cela semble raisonnable. C'est le genre de contenu que je m'attendrais à trouver dans une manchette, et le mot « header », ou en-tête, est souvent utilisé comme synonyme de manchette.

Il y a une différence cruciale entre l'élément `header` de l'HTML5 et l'usage répandu du mot « en-tête » ou « manchette ». Il n'y a généralement qu'une seule manchette sur une page, mais un document peut comporter plusieurs éléments `header`. Vous pouvez par exemple utiliser un élément `header` dans un élément `section`. En fait, c'est sans doute ce que vous *devriez* faire. La spécification décrit l'élément `section` comme « un regroupement de contenu thématique, *comprenant généralement un en-tête* ».

```
<section>
  <header>
    <h1>DOM Scripting</h1>
  </header>
  <p>Ce livre est destiné aux designers »
  plutôt qu'aux programmeurs.</p>
  <p>Par Jeremy Keith</p>
</section>
```

Un `header` apparaîtra normalement en haut d'un document ou d'une section, mais ce n'est pas obligatoire. Il est défini par son contenu (présentation ou aide à la navigation) plutôt que par sa position.

footer

Comme l'élément `header`, `footer` semble décrire une position, mais comme pour `header`, ce n'est pas le cas. L'élément `footer` contient plutôt des informations au sujet de l'élément qui le contient : auteur, informations de copyright, liens vers des contenus apparentés, etc.

Cela correspond assez bien à l'idée que les web designers se font du mot « footer ». La différence, c'est qu'on a l'habitude d'avoir un seul pied de page pour tout un document. L'HTML5 nous permet également d'en placer à l'intérieur des sections.

```
<section>
  <header>
    <h1>DOM Scripting</h1>
  </header>
  <p>Ce livre est destiné aux designers »
  plutôt qu'aux programmeurs.</p>
  <footer>
    <p>Par Jeremy Keith</p>
  </footer>
</section>
```

aside

Si l'élément `header` correspond au concept de la manchette, l'élément `aside` est une sorte d'encadré. Quand je dis « encadré », ce n'est pas une question de position. Il ne suffit pas que le contenu apparaisse à gauche ou à droite du contenu principal pour utiliser l'élément `aside`. Une fois de plus, c'est le contenu, et non la position, qui est important.

L'élément `aside` doit être utilisé pour du contenu indirectement apparenté. Si vous avez un morceau de contenu que vous considérez comme différent du contenu principal, l'élément `aside` est probablement le bon conteneur. Demandez-vous si le contenu de l'élément `aside` peut être retiré sans affecter le sens du contenu principal du document ou de la section.

Un bon exemple de contenu indirectement apparenté serait une citation extraite du document. Elle rend la lecture plus agréable, mais on peut la supprimer sans affecter la compréhension du contenu principal.

Gardez à l'esprit que si votre concept visuel prévoit de placer une partie du contenu dans un encadré, `aside` n'est pas

nécessairement le conteneur adéquat. Il est courant, par exemple, de placer la biographie de l'auteur dans un encadré. L'élément `footer` se prête mieux à ce genre de données, comme l'indique explicitement la spécification (FIG 5.01).

FIG 5.01 : Le texte « About the author » de cette capture d'écran doit être balisé avec `footer` et non `aside`.

Dans 90 % des cas, un header sera placé en haut de votre contenu, un footer à la fin, et un aside sur le côté. Mais méfiez-vous de cette apparente simplicité. Marchez sur des œufs et prenez garde aux 10 % qui restent.

nav

L'élément `nav` fait exactement ce qu'on attend de lui. Il contient des informations de navigation, généralement une liste de liens.

En fait, je ferais mieux de clarifier tout cela. L'élément `nav` est conçu pour les informations de navigation principales. Une simple liste de liens ne justifie pas l'utilisation de l'élément `nav`. En revanche, les liens permettant de naviguer sur un site doivent presque obligatoirement utiliser l'élément `nav`.

Bien souvent, un élément `nav` apparaîtra dans un élément `header`. Cela paraît logique si l'on considère que l'élément `header` peut être utilisé pour les « aides à la navigation ».

article

Il peut être utile de considérer `header`, `footer`, `nav` et `aside` comme des formes spécialisées de l'élément `section`. Une section est un morceau générique de contenu apparenté, tandis que les éléments `header`, `footer`, `nav` et `aside` sont des morceaux *spécifiques* de contenu apparenté.

L'élément `article` est un autre type de section spécialisé. On l'utilise pour le contenu apparenté autonome. Le problème étant de déterminer ce qui constitue un contenu « autonome ».

Demandez-vous si vous publieriez le contenu dans un flux RSS ou Atom. Si le contenu a toujours un sens dans ce contexte, `article` est probablement l'élément à utiliser. En fait, l'élément `article` est conçu spécifiquement pour la syndication.

Si vous utilisez un élément `time` dans un `article`, vous pouvez ajouter l'attribut booléen optionnel `pubdate` pour indiquer qu'il s'agit de la date de publication.

```
<article>
  <header>
    <h1>Critique de DOM Scripting</h1>
  </header>
  <p>Un vraie lueur d'espoir pour JavaScript au terme »
  d'une longue et parfois sombre traversée.</p>
```

```
    <footer>
      <p>Publié à
        <time datetime="2005-10-08T15:13" pubdate>
        15 h 13 le 8 octobre 2005
        </time>
      par Glenn Jones</p>
    </footer>
  </article>
```

S'il y a plus d'un élément `time` dans un article, un seul peut comporter l'attribut `pubdate`.

L'élément `article` est utile pour les billets de blog, les informations, les commentaires, les critiques et les fils de discussion. Il couvre exactement les mêmes cas d'utilisation que le microformat hAtom.

La spécification HTML5 va encore plus loin. Elle déclare également que l'élément `article` doit être utilisé pour les widgets autonomes : cours de la bourse, calculatrice, horloge, météo et autres. L'élément `article` tâche maintenant de couvrir les mêmes cas d'utilisation que les *web slices* de Microsoft (http://bkaprt.com/html5/8[1]).

Il me paraît très contre-intuitif d'appliquer un élément appelé « article » au concept des widgets. Mais après tout, les articles et les widgets sont tous deux des types de contenu publiables et autonomes.

Ce qui est plus problématique, c'est que les éléments `article` et `section` sont très similaires. Tout ce qui les sépare, c'est le mot « autonome ». Il serait facile de décider quel élément utiliser s'il existait des règles strictes. Au lieu de cela, tout est une question d'interprétation. Il peut y avoir plusieurs articles dans une section, il peut y avoir plusieurs sections dans un article, il peut même y avoir des sections dans les sections et des articles dans les articles. C'est à vous de décider quel est l'élément le plus approprié sémantiquement pour une situation donnée.

1. Adresse complète : http://www.ieaddons.com/en/webslices/

Un remède contre la div-ite ?

L'HTML5 apporte les quelques nouveaux éléments structurels décrits ci-dessus. Ceux-ci sont particulièrement utiles si vous construisez un site conventionnel, comme un blog. La plupart des blogs comprennent un en-tête suivi d'une série d'articles, avec du contenu parallèle dans un encadré et un pied de page (FIG 5.02).

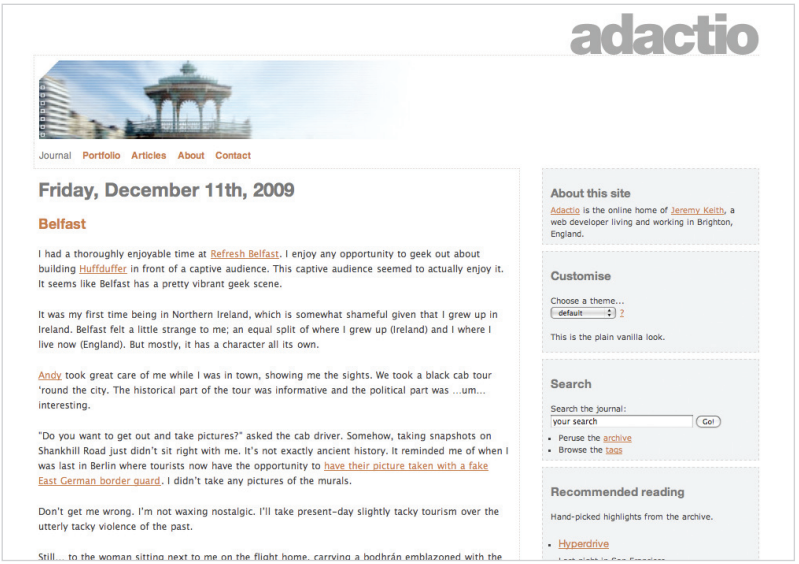


FIG 5.02 : Le blog de votre serviteur.

Vous pouvez maintenant remplacer certains de vos éléments `div` par des éléments structurels sémantiquement plus précis. Attention, n'en faites pas trop. Il y a fort à parier que si vous utilisez `div` aujourd'hui, vous utiliserez encore `div` demain. Ne changez pas vos éléments `div` pour des éléments HTML5 tout neufs, simplement parce que vous en avez la possibilité. Pensez au contenu.

Ces nouveaux éléments n'ont pas été créés dans le seul but de remplacer les éléments `div`. Ils offrent aux navigateurs web une toute nouvelle façon d'appréhender votre contenu.

MODÈLES DE CONTENU

Les versions précédentes des langages de balisage divisaient les éléments en deux catégories : les éléments de type en-ligne et les éléments de type blocs. L'HTML5 utilise une approche plus fine, répartissant les éléments dans un plus grand nombre de catégories.

Les éléments en-ligne ont maintenant un modèle de contenu de « sémantique de niveau texte ». De nombreux éléments de niveau bloc sont maintenant rassemblés sous la bannière des « contenus de regroupement » : paragraphes, éléments de liste, sections et ainsi de suite. Les formulaires ont leur propre modèle de contenu. Les images, l'audio, la vidéo et les canvas sont des « contenus intégrés ». Les nouveaux éléments structurels introduisent un modèle de contenu complètement nouveau appelé « contenu de segmentation » (*sectioning content*).

Contenu de segmentation

Il est possible de créer le plan d'un document HTML avec les éléments de titre `h1` à `h6`. Voyez plutôt cet exemple :

```
<h1>An Event Apart</h1>
<h2>Villes</h2>
<p>Rejoignez-nous en 2010 dans les villes suivantes.</p>
<h3>Seattle</h3>
<p>Suivez la route de brique jaune jusqu'à la ville
émeraude.</p>
<h3>Boston</h3>
<p>Beantown pour les intimes.</p>
<h3>Minneapolis</h3>
<p>La plus <em>belle</em>.</p>
<small>Logement non fourni.</small>
```

Ce qui nous donne le plan suivant :

- An Event Apart
  - Villes
    - Seattle
    - Boston
    - Minneapolis

Cela marche plutôt bien. Tout contenu placé à la suite d'un élément de titre est associé à ce titre.

Regardez maintenant le dernier élément `small`. Celui-ci devrait être associé au document dans son intégralité, mais un navigateur n'a aucun moyen de savoir que l'élément `small` ne doit pas dépendre de l'en-tête « Minneapolis ».

Le nouveau contenu de segmentation de l'HTML5 vous permet de délimiter explicitement le début et la fin du contenu apparenté :

```
<h1>An Event Apart</h1>
<section>
  <header>
    <h2>Villes</h2>
  </header>
  <p>Rejoignez-nous en 2010 dans les villes suivantes.</p>
  <h3>Seattle</h3>
  <p>Suivez la route de brique jaune.</p>
  <h3>Boston</h3>
  <p>Beantown pour les intimes.</p>
  <h3>Minneapolis</h3>
  <p>La plus <em>belle</em>.</p>
</section>
<small>Logement non fourni.</small>
```

Il est maintenant clair que l'élément `small` dépend de l'en-tête « An Event Apart » et non de l'en-tête « Minneapolis ».

Il est possible de subdiviser encore plus ce contenu, en plaçant chaque ville dans sa propre section :

```
<h1>An Event Apart</h1>
<section>
  <header>
    <h2>Villes</h2>
  </header>
  <p>Rejoignez-nous en 2010 dans les villes suivantes.</p>
  <section>
    <header>
      <h3>Seattle</h3>
    </header>
    <p>Suivez la route de brique jaune.</p>
  </section>
  <section>
    <header>
      <h3>Boston</h3>
    </header>
    <p>Beantown pour les intimes.</p>
  </section>
  <section>
    <header>
      <h3>Minneapolis</h3>
    </header>
    <p>La plus <em>belle</em>.</p>
  </section>
</section>
<small>Logement non fourni.</small>
```

Ce qui nous donne le même plan :

- An Event Apart
  - Villes
    - Seattle
    - Boston
    - Minneapolis

L'algorithme de calcul du plan

Jusqu'ici, le nouveau contenu de segmentation ne nous aide pas beaucoup plus que les versions précédentes de l'HTML. La nouveauté de l'HTML5, c'est que chaque section possède son propre plan autonome. Cela veut dire que vous n'avez plus besoin de garder le fil du niveau de titre à utiliser. Vous pouvez partir de `h1` à chaque fois :

```
<h1>An Event Apart</h1>
<section>
  <header>
    <h1>Villes</h1>
  </header>
  <p>Rejoignez-nous en 2010 dans les villes suivantes.</p>
  <section>
    <header>
      <h1>Seattle</h1>
    </header>
    <p>Suivez la route de brique jaune.</p>
  </section>
  <section>
    <header>
      <h1>Boston</h1>
    </header>
    <p>Beantown pour les intimes.</p>
  </section>
  <section>
    <header>
      <h1>Minneapolis</h1>
    </header>
    <p>La plus <em>belle</em>.</p>
  </section>
</section>
<small>Logement non fourni.</small>
```

Les versions précédentes de l'HTML auraient produit un plan erroné :

- An Event Apart
- Villes
- Seattle
- Boston
- Minneapolis

En HTML5, le plan est correct :

- An Event Apart
  - Villes
    - Seattle
    - Boston
    - Minneapolis

hgroup

Parfois, on voudra utiliser un élément de titre sans voir son contenu apparaître dans le plan du document. C'est ce que vous permet de faire l'élément `hgroup` :

```
<hgroup>
  <h1>An Event Apart</h1>
  <h2>Pour ceux qui font le web</h2>
</hgroup>
```

Dans ce cas, le titre de niveau deux (« Pour ceux qui font le web ») n'est en fait qu'un slogan. Dans un élément `hgroup`, seul le premier en-tête fera partie du plan. Celui-ci n'est pas nécessairement un `h1` :

```
<hgroup>
  <h3>DOM Scripting</h3>
  <h4>Web design avec JavaScript »
  et le Document Object Model</h4>
</hgroup>
```

Racines de segmentation

Certains éléments sont invisibles sur le plan généré. En d'autres termes, quel que soit le nombre de titres que contiennent ces éléments, ceux-ci n'apparaîtront pas dans le plan du document

Les éléments `blockquote`, `fieldset` et `td` ne sont pas pris en compte par l'algorithme de calcul du plan. Ces éléments sont appelés « racines de segmentation » (*sectioning roots*), à ne pas confondre avec le contenu de segmentation.

Portabilité

Puisque chaque contenu de segmentation génère son propre plan, vous n'avez plus à vous contenter des éléments `h1` à `h6`. Il n'y a pas de limite au nombre de niveaux de titres, mais surtout, vous pouvez commencer à penser votre contenu de manière vraiment modulaire.

Supposons que je veuille publier un billet sur mon blog appelé « Sandwich au fromage ». Avant l'HTML5, il m'aurait fallu connaître le contexte du billet afin de décider du niveau d'en-tête à utiliser pour le titre. Si le billet est sur le page d'accueil, il apparaît alors après un élément `h1` contenant le titre de mon blog :

```
<h1>Mon blog génial</h1>
<h2><a href="fromage.html">Sandwich au fromage</a></h2>
<p>Mon chat a mangé un sandwich au fromage.</p>
```

Mais si je publie le billet sur sa propre page, je veux que le titre soit un titre de niveau un :

```
<h1>Sandwich au fromage</h1>
<p>Mon chat a mangé un sandwich au fromage.</p>
```

En HTML5, je n'ai pas à me soucier du niveau de titre à utiliser. Il me faut simplement utiliser un contenu de segmentation, dans ce cas, un élément `article` :

```
<article>
  <h1>Sandwich au fromage</h1>
  <p>Mon chat a mangé un sandwich au fromage.</p>
</article>
```

Maintenant, le contenu est vraiment portable. Le fait qu'il apparaisse sur la page d'accueil ou sur sa propre page n'a aucune importance :

```
<h1>Mon blog génial</h1>
<article>
  <h1>Sandwich au fromage</h1>
  <p>Mon chat a mangé un sandwich au fromage.</p>
</article>
```

Le nouvel algorithme de calcul du plan de l'HTML5 produit le bon résultat :

- Mon blog génial
  - Sandwich au fromage

Scoped

Le fait que chaque contenu de segmentation dispose de son propre plan est idéal pour Ajax. Une fois de plus, l'HTML5 affiche l'ambition d'une spécification conçue pour les applications web.

Il peut être problématique de transférer un bout de contenu d'un document à un autre. Les règles CSS s'appliquant au document parent s'appliqueront également au contenu inséré. On touche là à l'un des défis majeurs de la distribution des widgets sur le Web.

L'HTML5 offre une solution à ce problème sous la forme de l'attribut `scoped`, qui peut s'appliquer à un élément `style`. Tout

style déclaré dans cet élément ne s'appliquera qu'au contenu de segmentation qui le contient :

```
<h1>Mon blog génial</h1>
<article>
  <style scoped>
    h1 { font-size: 75% }
  </style>
  <h1>Sandwich au fromage</h1>
  <p>Mon chat a mangé un sandwich au fromage.</p>
</article>
```

Dans cet exemple, seul le deuxième élément `h1` aura une valeur `font-size` de 75 %. Voilà pour la théorie : aucun navigateur ne supporte encore l'attribut `scoped`.

Et c'est bien le problème. Avant de pouvoir commencer à utiliser les nouveautés de l'HTML5, vous devrez réfléchir à la compatibilité des navigateurs. Je connais quelques stratégies qui vous permettront de commencer à utiliser l'HTML5, quelle que soit la compatibilité des navigateurs. Dans le dernier chapitre, j'aimerais en partager quelques-unes avec vous.

6 UTILISER L'HTML5 AUJOURD'HUI

Si vous voulez commencer à utiliser les nouveaux éléments structurels de l'HTML5 dès aujourd'hui, rien ne vous en empêche. La plupart des navigateurs vous permettront de styler ces nouveaux éléments. Non pas parce que ces navigateurs supportent activement ces éléments, mais parce que la plupart des navigateurs vous permettent d'utiliser et de styler n'importe quel élément de votre création.

STYLE

Les navigateurs n'appliqueront pas de style par défaut aux nouveaux éléments. Il vous faudra donc, au minimum, déclarer que les nouveaux éléments structurels doivent imposer un saut de ligne :

```
section, article, header, footer, nav, aside, hgroup {
  display: block;
}
```

Cela suffit pour la plupart des navigateurs, mais Internet Explorer a des besoins spéciaux. Il refuse catégoriquement de reconnaître les nouveaux éléments, à moins qu'un exemplaire de chaque élément n'ait été créé au préalable à l'aide de JavaScript, comme ceci :

```
document.createElement('section');
```

Remy Sharp, le génie du JavaScript, a écrit un petit script très pratique qui génère tous les nouveaux éléments HTML5. Chargez ce script avec un commentaire conditionnel afin qu'il ne soit utilisé que par le chétif Internet Explorer :

```
<!--[if IE]>
  <script src= »
  "http://html5shiv.googlecode.com/svn/trunk/html5.js">
  </script>
<![endif]-->
```

Vous pouvez maintenant styler les nouveaux éléments à votre aise.

En-têtes

Les navigateurs ne supportent pas encore le nouvel algorithme de calcul du plan de l'HTML5, mais vous pouvez quand même commencer à utiliser les niveaux de titre supplémentaires mis à votre disposition.

Geoffrey Sneddon a mis en ligne un outil, bien commode lui aussi, qui génère un plan comme spécifié en HTML5 (http://bkapart.com/html5/9[1]).

Si vous suivez le conseil de la spécification HTML5, en partant de `h1` dans chaque contenu de segmentation, vos règles CSS pourraient bien vite se compliquer :

1. Adresse complète : http://gsnedders.html5.org/outliner

```
h1 {
  font-size: 2.4em;
}
h2,
  section h1, article h1, aside h1 {
    font-size: 1.8em;
}
h3,
  section h2, article h2, aside h2,
  section section h1, section article h1, section aside h1,
  article section h1, article article h1, article aside h1,
  aside section h1, aside article h1, aside aside h1 {
    font-size: 1.6em;
}
```

Voilà pour les trois premiers niveaux seulement, et cela n'inclut même pas toutes les combinaisons d'en-têtes potentielles dans un contenu de segmentation.

Heureusement, l'algorithme de calcul du plan de l'HTML5 est plutôt flexible. Si vous souhaitez utiliser les niveaux de titres à l'ancienne, le plan n'en sera pas affecté.

ARIA

Les nouveaux éléments structurels de l'HTML5 seront très utiles pour les technologies d'accessibilité. Au lieu de créer des liens d'évitement (*skip navigation*), il nous suffit d'utiliser l'élément `nav` correctement pour permettre aux utilisateurs de lecteurs d'écran de passer d'une section à une autre sans que nous ayons à fournir un lien explicite.

C'est en tout cas ce qui est prévu. Il faut pour l'instant faire avec les technologies que les navigateurs et les lecteurs d'écran supportent déjà.

Par chance, il existe déjà une compatibilité excellente avec ARIA (Accessible Rich Internet Applications).

Au maximum de ses capacités, ARIA permet aux technologies d'accessibilité de participer à des interactions Ajax quasiment avant-gardistes. Plus simplement, ARIA nous permet d'enrichir la sémantique de nos documents.

L'unité ARIA de base est l'attribut `role`. Vous pouvez ajouter `role="search"` à votre formulaire de recherche, `role="banner"` à votre manchette et `role="contentinfo"` à votre pied de page. Vous trouverez la liste complète des valeurs dans la spécification ARIA, à l'adresse http://bkapart.com/html5/10[1].

Vous pouvez également utiliser l'attribut `role` en HTML 4.01, en XHTML 1.0 ou avec tout autre langage de balisage, mais votre document ne pourra plus être validé, à moins que vous ne créiez un doctype personnalisé, ce qui est extrêmement fastidieux.

En revanche, les rôles ARIA font partie de la spécification HTML5, si bien que vous pouvez avoir le beurre et, à défaut de l'argent du beurre, le sourire du validateur.

Vous pouvez également utiliser la sémantique ajoutée de l'attribut `role` pour gérer les styles. Le sélecteur d'attribut est votre ami. Ces sélecteurs vous permettent de distinguer les en-têtes et les pieds de page d'un document de ceux d'un contenu de segmentation :

```
header[role="banner"] { }
footer[role="contentinfo"] { }
```

VALIDATION

Utilisé à bon escient, un validateur est un outil très puissant pour un web designer. Autrement, c'est un moyen facile pour des nerds snobinards de se moquer du travail des autres.

1. Adresse complète : http://www.w3.org/TR/wai-aria/roles#role_definitions

Henri Sivonen a créé un validateur HTML5 complet à l'adresse http://validator.nu/.

Vous n'avez même pas besoin de mettre à jour l'adresse du validateur W3C (http://validator.w3.org/) dans vos favoris. Celui-ci utilise également le parseur d'Henri dès qu'il détecte le doctype HTML5.

DÉTECTION DE FONCTIONNALITÉS

Si vous voulez commencer à utiliser certains des types de champs avancés de l'HTML5, il vous faudra une méthode pour tester la compatibilité des navigateurs afin de fournir des alternatives en JavaScript.

Modernizr est un fichier JavaScript très utile qui détecte le support des types de champs et des éléments `audio`, `video` et `canvas` (http://www.modernizr.com/).

Ce script crée un objet en JavaScript appelé Modernizr. En interrogeant les propriétés de cet objet, vous saurez si le navigateur supporte ou non tel type de champ :

```
if (!Modernizr.inputtypes.color) {
  // Placez ici la solution de secours en JavaScript.
}
```

Modernizr fait également un tour de passe-passe qui vous permet de styler les nouveaux éléments structurels sous Internet Explorer. Ainsi, si vous utilisez Modernizr, pas besoin d'utiliser le script de Remy.

CHOISISSEZ VOTRE STRATÉGIE

Avec l'HTML5, il ne tient qu'à vous d'être ambitieux, ou prudent.

Au grand minimum, vous pouvez prendre vos documents HTML ou XHTML existants et changer le doctype en :

```
<!DOCTYPE html>
```

Vous venez de mettre le doigt dans l'engrenage. Tant qu'à faire, vous pouvez également utiliser les rôles ARIA. Qu'avez-vous à perdre ?

Si le fait d'utiliser les nouveaux éléments structurels vous angoisse, vous pouvez tout de même vous habituer à la nouvelle sémantique en vous aidant des noms de classes comme flotteurs :

```
<div class="section">
  <div class="header">
    <h1>Hello world !</h1>
  </div><!-- /.header -->
</div><!-- /.section -->
```

Par la suite, quand vous vous sentirez plus à l'aise avec les nouveaux éléments HTML5, vous pourrez remplacer ces éléments `div` et ces noms de classes par les éléments structurels correspondants.

Bien qu'il soit encore trop tôt pour utiliser les types de champs les plus avancés tels que `date`, `range` et `color`, il n'y a aucun mal à utiliser `search`, `url`, `email` et d'autres types de champs simples. Souvenez-vous, les navigateurs qui ne reconnaissent pas ces valeurs traiteront simplement le champ comme s'il était de type `text`.

Si vous avez l'âme d'un aventurier, vous pouvez commencer à jouer avec les éléments `audio`, `video` et `canvas`. Sans être encore prêts pour le grand jour, ils peuvent être amusants à expérimenter sur votre site personnel.

Ressources

J'écris souvent au sujet de l'HTML5 sur mon site personnel :
http://adactio.com/journal/tag/html5

Je ne suis pas le seul à m'emballer. Le mythique Bruce Lawson partage également ses réflexions :
http://brucelawson.co.uk/category/html5

Bruce est un des nombreux contributeurs de HTML5 Doctor, une excellente ressource communautaire comprenant de nombreux articles intéressants :
http://html5doctor.com

Si vous voulez vous attaquer aux aspects plus complexes de l'HTML5, Remy Sharp repousse les limites :
http://html5demos.com

Mark Pilgrim a écrit un livre très complet appelé *Dive Into HTML5* (en anglais). Achetez-le chez O'Reilly ou lisez-le en ligne :
http://diveintohtml5.org

Pour toutes les fois où vous devrez aller droit à la source, gardez la spécification HTML5 à portée de main :
http://whatwg.org/html5

La spécification HTML5 comprend beaucoup d'informations destinées aux créateurs de navigateurs. Le W3C héberge une version à jour de la spécification spécifiquement pour les auteurs :
http://www.w3.org/TR/html-markup

IMPLIQUEZ-VOUS

En vous engageant dans l'aventure HTML5, vous serez peut-être dérouté par certaines parties de la spécification. Ce n'est pas grave, c'est même une excellente chose : vos impressions comptent.

Des personnes très intelligentes travaillent sur l'HTML5, mais les web designers sont en minorité. Votre point de vue est donc grandement recherché.

Vous pouvez rejoindre le groupe de travail HTML du W3C en tant qu'expert public invité (oubliez cette histoire kafkaïenne d'auto-invitation), mais je ne vous le recommande pas. La liste de diffusion associée engendre un trafic considérable, essentiellement pour des histoires de politiques et de procédure.

La liste de diffusion du WHATWG est le lieu à visiter si vous voulez véritablement débattre de la spécification HTML5 : http://www.whatwg.org/mailing-list#specs

Il existe également un canal IRC. Parfois, on aime bien se retrouver entre soi : irc://irc.freenode.org/whatwg

Ne soyez pas timide. Le canal IRC est le lieu idéal pour poser des questions et obtenir les réponses de Ian Hickson, d'Anne van Kesteren, de Lachlan Hunt et d'autres membres du WHATWG.

LE FUTUR

J'espère que cette petite balade dans le monde de l'HTML5 vous a donné l'envie d'explorer cette technologie passionnante. J'espère également que vous rapporterez les fruits de votre exploration au WHATWG.

L'HTML est l'outil le plus important qu'un Web designer puisse manier. Sans le balisage, le web n'existerait pas. Je trouve à la fois remarquable et merveilleux que *tout le monde* puisse contribuer à l'évolution de cette technologie des plus vitales. Chaque fois que vous créez un site web, vous contribuez à l'héritage culturel commun à tous les êtres humains. En choisissant l'HTML5, vous contribuez aussi au futur.

INDEX

Q

R

S

T

U

V

W

X

A BOOK APART :
LES LIVRES DE CEUX QUI FONT LE WEB

Le web design est une affaire de maîtrise multidisciplinaire et de haute précision. C'est justement l'idée qui se reflète dans notre nouvelle collection de petits livres, destinée à tous ceux qui créent des sites Web.

Les ouvrages de la collection « A Book Apart » sont des études approfondies et minutieusement éditées sur des sujets précis. Nous avons le plaisir de lancer cette collection avec *HTML5 pour les web designers* de Jeremy Keith.

À PROPOS DE L'AUTEUR

Auteur de deux autres ouvrages *DOM Scripting* et *Bulletproof Ajax*, Jeremy Keith est un développeur Web irlandais qui vit à Brighton en Angleterre, où il collabore avec le cabinet de conseil Web Clearleft. Son site personnel est adactio.com et son dernier projet, Huffduffer, un service qui permet de créer des podcasts à partir de sons trouvés sur le Web. Quand il ne crée pas des sites Web, Jeremy joue du bouzouki dans le groupe Salter Cane.